Устройство USB и FireWire.

USB and FireWire design

Alexandre Rusev
rusev@asoft.ru
http://www.asoft.ru

Эпиграф:
"И может быть мы сразу друг друга поймём -
Если у нас один и тот же разъём."

Прочесть это нетленное произведение полностью можно сдесь:
http://lyrix.ru/songs/?id=182314
;))

Вместо аннотации ;)

В настоящее время на рынке появляется все больше недорогих электронных устройств бытового и профессионального назначения, которые используют для связи с компьютером другими устройствами последовательные интерфейсы FireWire и USB, приходящие на смену устаревшим RS232 и LPT.
Среди них устройства передачи и обработки видеоизображений, принтеры, сканеры, мобильные телефоны, мыши, клавиатуры и.т.п.
В ряде случаев последовательные шины так же являются хорошей заменой распространенным параллельным шинам таким как PCI и ISA.
Все больше разработчиков прикладного ПО, драйверов устройств для различных ОС а так же разработчиков электронных устройств сталкиваются с необходимостью использования или реализации протоколов этих шин.
Данная публикация посвящена изложению архитектуры а так же основных особенностей работы и использования шин FireWire и USB. На протяжении всего изложения акцентируется внимание на сходствах и различиях этих двух шин.

Содержание

Провода и разъемы
Кабели и разъемы USB.
Кабели и разъемы FireWire.

Модель передачи данных
Основные способы передачи данных (асинхронный и изохронный)
Назначения способов передачи данных, устройства их использующие

Управление шиной
“Централизованное” управление шиной USB
“Самоорганизующееся” управление шиной FireWire (IEEE1394)

Определение топологии шины
USB
FireWire
Tree Identification.
Self identification

Передача пакетов
USB.
Таймслоты, совместимость стандартов, транзакции.
Структура и типы используемых пакетов
Split-транзакции
Описание транзакций для разных типов передачи данных USB (bulk, isochron, control, interrupt)
Bulk транзакции
Control транзакции
Interrupt транзакции
Isochron транзакции
FireWire.
Уровни протокола FireWire
Split-транзакции

Электрические характеристики, передача сигналов
USB.
FireWire.

Управление питанием
USB.
FireWire.

Работа программы (или драйвера) с шиной
USB.
FireWire.
Unit registers
PHY Registers и Configuration ROM
Configuration ROM

Аппаратные решения для устройств USB и FireWire

Поддержка FireWire, USB различными платформами

Исходные тексты программ и другие источники информации в интернет

Другие последовательные шины и беспроводные каналы связи.
Предшествующие реализации.
Современные последовательные проводные шины.
Шина I2C.
Шина CAN.

Беспроводные сети и каналы передачи.
BlueTooth.
WLAN.

Литература.

Введение

Назначение USB и FireWire

Списки потребности, которые по мнению их разработчиков должны были решить USB и FireWire, во многом пересекаются и состоят в следующем:

1. Простота в использовании
2. Универсальность по отношению к различным применениям (например возможность заменить функции COM и LPT портов в PC)
3. Масштабируемость по количеству устройств
4. Невысокая стоимость
5. Возможность подключения большого количества устройств и масштабируемость с использованием HUB-ов
6. Возможность “горячего” подключения и отключения устройств
7. Возможность относить на достаточно большие расстояния (по сравнению с такими параллельными шинами как ISA,PCI,AGP) устройств соединенных при помощи шины.
8. Небольшое количество соединительных проводов (тонкие кабели) по сравнению с наиболее распространенными компьютерными последовательными шинами (RS232,RS485,LPT)
9. Поддержка высоких скоростей передачи данных, достаточной для передачи видео и аудио.
10. Возможность передачи данных в реальном времени (с фиксированной максимальной временной задержкой), как это необходимо для передачи звука и изображения
11. Возможность автоматической настройки ПО при подключении новых устройств
12. Для FireWire еще и возможность работы соединенных устройств в отсутствии основного компьютера (хоста)
13. Для FireWire возможность одновременного получения потока данных несколькими устройствами (broadcast)
14. USB по стоимости должна была подходить в качестве замены последовательным и параллельным портам для компьютеров следующего поколения.

Зачем нам новые шины (или разъемы на задней панели) вместо старых?

“Зачем нам новые шины вместо старых?” - этот вопрос уместно задать в первую очередь в отношении USB, так как именно USB по замыслу предназначается в качестве замены имеющимся в компьютерах COM и LPT портам. В отношении же FireWire можно сразу же заметить следующее, что хотя FireWire и предоставляет нам некоторые чисто компьютерные сервисы (такие, например, как подключение к PC скоростных внешних накопителей или сеть TCP/IP по FireWire в ОС Linux) она все же пришла в сферу персональных компьютеров из области цифровой передачи изображения, где традиционно используется уже на протяжении некоторого времени (например видеокамерами Sony).

Сегодня компьютеры стали существенно быстрее благодаря чему пользователь получил возможность включить свой PC в процесс взаимодействия с цифровой видеозаписывающей аппаратурой. Особое развития получили компьютерные средства мультимедиа и FireWire стала востребована в мире персональных компьютеров как никогда ранее. В силу особенностей своего происхождения FireWire унаследовала способность соединять устройства и в отсутствии центрального компьютера, что делает ее еще более универсальной чем USB, правда при этом и более дорогой.

Что же касается USB, то она тоже предоставляет возможность передачи данных. Достаточно быстрой для подключения r PC различных мультимедийных устройств. При этом более низкая стоимость делает USB в ряде случаев еще более привлекательной. И все же зачем нам USB? Известно что через порты USB1.0 может передаваться до 12Mbit/sec в то время как современные варианты LPT портов способны передавать до 24Mbit/sec и только последовательные порты COM отстают от USB1.0 примерно в 100 раз, их скорость обычно не достигает более 155KBit/sec. Пожалуй если мы используем только принтер мышь и модем USB1.0 не даст нам существенных преимуществ. В тех же случаях когда требуется подключить к компьютеру пул из 10 модемов, долгое время такая задача решалась при помощи мультипортовой платы, вставляемой в ISA или PCI слот. Безусловно USB1.0 является более элегантным решением в плане масштабируемости и универсальности, поэтому в частности и производители видеокамер, предназначенных для подключения к PC перешли с использования параллельного порта или собственной платы для подключения камеры к использованию USB и FireWire. Очевидно что с развитием элементной базы аппаратного обеспечения персональных компьютеров USB1.0 потихоньку вошла в жизнь пользователей PC. Во всех перечисленных случаях она имеет неоспоримые преимущества перед последовательными шинами предыдущего поколения, хотя и не предоставляет каких=то особых выдающихся возможностей по сравнению с ними.

Ситуация в корне изменилась с появлением USB2.0, которая по скорости передачи оставила далеко позади COM и LPT. К сожалению еще не так много имеется недорогих устройств различного назначения с USB2.0, но прогресс как всегда неостановим и USB потихоньку “берет свое”, занимая место устаревающих последовательных и параллельных портов.

Автоматический поиск драйверов устройств Plug & Play

USB и FireWire изначально разрабатывались с учетом того, что программное обеспечение ОС должно самостоятельно определять какой драйвер следует использовать для работы с каждым вновь подключаемым устройством. Это особенно важно с учетом того, что устройства могут подключаться и отключаться находу к различным портам компьютера и в такой ситуации прользователю было бы слишком утомительно каждый раз сообщать системе что скажем к порту номе 8 подключено такое-то устройство.

Для того, чтобы в полной мере реализовать такую возможность каждое устройство должно иметь внутри некоторый уникальный идентификатор (идентификатор производителя и идентификатор самого устройства:Vendor ID и Product ID). Для того, чтобы идентификаторы случайно не повторились для двух различных устройств, сделанных разными производителями, они должны выдаваться некоторым центральным органом по регистрации таких идентификаторов. Соответствующим стандарту может считаться только лишь устройство, которое имеет зарегистрированные идентификаторы. На практике именно это имеет место. Получение идентификатора коммерчеси доступно любому производителю электронных устройств.

Обычно компьютеры с ОС Windows или Linux способны самостоятельно обнаруживать устройства, такие например как Web-камеры с интерфейсом USB и использовать соответствующий им драйвер устройства.

В Linux так же возможно использование USB портов и устройств подключенных к ним прикладной программой, как это всегда было с последовательными портами RS232. К сожалению в Windows в настоящее время использование USB является существенно более сложной задачей, требующей написания системного драйвера.

Для тех же кому интересна работа системных драйверов с USB устройствами в настоящее время можно так же порекомендовать использование Linux в связи с тем, что там имеется достаточно большое количество драйверов устройств с исходными текстами. В качестве примера для изучения можно порекомендовать драйвер для Web-камеры Logitech QuickCam, который по мнению автора является достаточно простым и вместе с тем иллюстрирует все основные особенности работы системного драйвера с USB устройством.

Версии стандартов FireWire и USB

С момента своего введения оба стандарта претерпели развитие в частности в сторону увеличения скорости передачи, поэтому в первую очередь речь пойдет именно о различиях в скорости.

Традиционно скорость передачи по последовательным линиям и шинам (как например когда речь идет о модемах) скорость передачи данных измеряется в единицах битов в секунду, а не в единицах байтов, как это делается для параллельных шин таких как ISA,PCI или AGP. Поэтому мы так же приводим скорости в мегабитах.

Следует иметь в виду что в силу устройства обеих шин максимальнодостижимая скорость однонаправленного канала передачи данных между двумя устройствами на шине несколько ниже приводимой здесь общей пропускной способности шины в мегабитах. Это объясняется тем, что шина не позволяет зарезервировать всю свою полосу пропускания под единственный канал передачи.

К одной шине могут оказаться могут оказаться подсоединенными устройства, удвлетворяющие разным версиям стандарта (и соответственно имеющие разные максимальные скорости передачи) таким образом что для передачи данных от одного устройства к другому может потребоваться их прохождение через узлы с различной скоростью. При этом шина в целом сохранит работоспособность, однако ограничения, однако ограничения, связанные со скоростью передачи могут быть наложены на работу отдельных устройств.

Поэтому при соединении разнотипных устройств в ряде случаев следует уделять особое внимание топологии их соединения.

Версии USB

В настоящее время существуют 2 версии стандарта USB: USB1.0 и USB2.0. USB2.0 обратносовместим с USB1.0. Максимальная скорость передачи данных USB1.0 равна 12Mbit/sec, USB2.0 обеспечивает скорость передачи примерно в 20 раз больше 480Mb/sec.

Существует еще тип так называемых медленных USB устройств, которые могут работать на скоростях лишь до 1.5Mb/sec. Шина обнаруживает такие устройства и осуществляет процесс взаимодействия с ними несколько подругому чем с остальными устройствами. Такие устройства так же имеют существенные ограничения по способам передачи данных.

Обе версии стандарта определяют хост (центральный компьютер) как элемент необходимый для работы шины в целом.

Версии FireWire (стандарта IEEE1394)

Шина FireWire специфицируется стандартом IEEE 1394-1995 (принятым в 1995 году) и его дальнейшим развитием версиями стандарта : IEEE 1394a и IEEE 1394B. Стандарт IEEE 1394a на сегодняшний день по-видимому является наиболее распространенным.

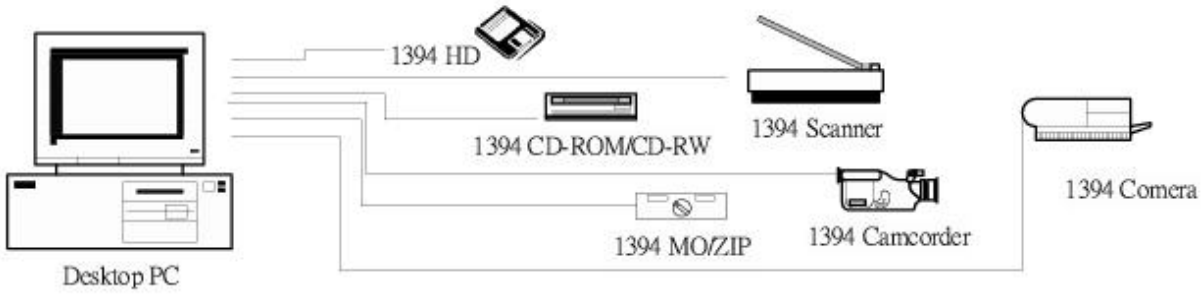
Скорости передачи данных, определяемые стандартами IEEE 1394-1995 и IEEE 1394a – это 100Mb/sec, 200Mb/sec, 400Mb/sec. Каждая из них может использоваться устройствами, подключенными к шине. Все FireWire устройства должны поддерживать работу на минимальной скорости 100Mb/sec.

В IEEE 1394B получили развитие еще большие скорости передачи: 800Mb/sec, 1.6Gb/sec, 3.2Gb/sec.

Во всех вариантах стандарта шина может функционировать в отсутствии выделенного узла (компьютера).

Примеры устройств

Рисунок (Различные типы устройств FireWire, подключаемых к компьютеру)



В этом пункте мы постараемся дать небольшой обзорустройств, использующих USB и FireWire.

Клавиатуры и мыши – как медленные устройства используют USB (как правило USB1.0)
WEB-камеры. Примером распространенной USB WEB-камеры может служить Logitech QuickCam. Камера способна передавать 12-15 кадров низкого разрешения (320x200) в секунду и вполне подходит для проведения видеоконференций через Интернет с использованием таких программ как NetMeeting, KMeeting или подобных. Однако качество изображения, получаемого при помощи этой камеры не слишком высокое, при уличном освещении (в солнечный день) изображение оказывается достаточно неразборчивым так как устройство предназначено исключительно для домашнего использования в условиях когда в кадр попадают предметы примерно одинаковой освещенности. Подстройка чувствительности камеры производится программным путем при помощи усреднения

яркости пикселов изображения, что может отнимать достаточно много ресурсов процессора. Баланс белого тоже оставляет желать лучшего по сравнению с любительскими цифровыми фотокамерами. Все названные недостатки относятся к большинству WEB-камер, коотрым не требуется высокое качество в силу того, что передача изображения через интернет происходит на крайне низких скоростях и поэтому передается достаточно статическое изображение довольно малого размера. Тем не менее Logitech QuickCam можно порекомендовать для большинства практических применений, связанных с Интернет. Программная поддержка этой камеры доступна для большинства операционных систем, включая все версии Wnidows и Linux.

Другим примером WEB-камеры является FireWire камера Pyro WebCam, которая существенно ближе к видеокамерам для любительской съемки, чем камера QuickCam. Pyro способна передавать уже до 30 кадров в секунду при чем с уже достаточно неплохим разрешением. Однако Pyro WebCam уже существенно дороже (примерно в 2 раза)

Рисунок (Logitech QuickCam Pro 4000 с интерфейсом USB)

Рисунок (WEB-камера Pyro с интерфейсом FireWire)

Видеокамеры: особо следует выделить видеокамеры с интерфейсом FireWire, к каковым относятся например некоторые модели производства Sony, многие из них уже являются высококачественными скоростными устройствами для видеосъемки. Некоторые способны самостоятельно записывать видеоизображения. FireWire традиционно используется Sony именно для этих целей. Как правило – это уже достаточно дорогостоящие устройства, использование которых вместе с компьютером получило широкое распространение относительно недавно.

Устройства Desktop Video, **TV-тюнеры**: к этой категории относятся многочисленные устройства для преобразования изображения между аналоговыми (такими как телевизионные форматы PAL,SECAM,NTSC) и цифровыми форматами изображения и звука. Некоторые их этих устройсв поддерживают так же введение визуальных эффектов видеоизображения на аппраратном уровне и предназначены для нелинейного монтажа видео. Эти устроства так же традиционно используют FireWire и скоростную шину USB2.0.

Mass Storage устройства к таковым относятся подключаемые к компьютеру при помощи USB и FireWire устройства CDROM , DVDROM, жесткие и Flash диски. Пожалуй здесь нечего добавить как и для следующего класса устройств:

Принтеры и сканеры – обычно используют USB.

Мобильные телефоны, компьютеры наладонники (Palm, Qompaque, Cassio и.т.д.), цифровые фотоаппараты - все эти устройства используют USB для обмена пользовательскими данными с компьютером.

Специализированные устройства сбора данных – для примера скоростные платы аналоговоцифровых и цифроаналоговых преобразователей производства Texas Instruments, SunDance или московской фирмы ЗАО “Инструментальные системы” (www.insys.ru) используют для передачи данных FireWire и USB2.0. Применение этих шин в данном случае более чем оправдано поскольку устройства сбора данных (АЦП/ЦАП), применяемые в связи, технике промышленных и научных измерений часто необходим для предотвращения помех расположить как можно дальше от компьютера и как можно ближе к датчику, показания которого снимаютя устройством. При этом нередко требуются достаточно высокие скорости передачи данных, такие как например в случае 14-битного АЦП, работающего с частотой 100МГц. Более подробно с этими устройствами можно ознакомиься на сайтах названных производителей.

Устройства для соединения двух компьютеров по USB. Так же как раньше было популярно использование так называемого нуль-модемного кабеля для этих целей, в настоящее время используются специальные USB устройства, выполняющие те же функции.

Сети FireWire – некоторые ОС ,такие например как Linux, позволяют организовать работу локальной сети TCP/IP при помощи соединений FireWire.

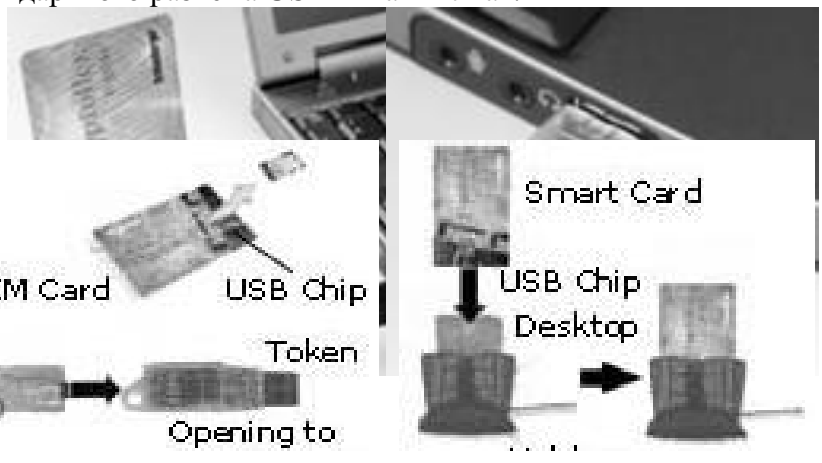
Рисунок (Ридер смарткарт ACR30U с интерфейсом USB)

Устройства чтения/записи смарткарт – в последнее время получают распространение чиповые карты и устройства чтения/записи таких карт (ридеры). Многие производители выпускают версии своих устройств предназначенные для использования последовательного порта RS232 а так же USB1.0. Примерами таких ридеров могут служить ACR30U производства (Advanced Card Systems), GCR410(производства Gemplus), ряд устройств производства Schumberger, ридеры Towitoko (распространяемые в Росии под маркой РусКард) .

Электронные ключи Aladdin – на смену знакомым пользователям 1С бухгагалтерии приходят миниатюрные ключи eToken, подсоединяемые через USB1.0. Ключи eToken фактически сочетают в себе смарткарту вместе ридером.

Рисунок (Aladdin eToken)

Существуют так же с**маркарты, которые могут при помощи примитивного переходника подключаться непосредственно к шине USB**, в частности чиповая карта eGate производства Schlumberger. В этом случае считыватель представляет из себя просто держатель и колодку с разводкой проводов от контактов смаркарты к контактам стандартного разъема USB типа “вилка”.



Отдельным классом устройств являются HUB-ы, эти устройства служат не только разветвителями для увеличения количества одновременно подключенных устройств, но и могут так же обеспечивать дополнительные источники питания для этих устройств.
Очевидно этим не исчерпывается все существующее на сегодняшний деь многообразие устройств, но мы надеемся что приведенный список примеров достаточен чтобы представить круг возможных применений USB и FireWire.

В настоящее время большинство материнских плат PC имеют встроенный контроллер USB, но не имеют такового для FireWire, поэтому для FireWire как правило приходится использовать дополнительный адаптор, подключаемый к шине PCI. Если требуется использовать FireWire с нотбуком, то правильнее выбрать модель нотбука, имеющую встроенный порт IEEE1394. Очень многие современные нотбуки имеют такие порты. Кроме того что носимые компьютеры как правило имеют не очень много возможностей для подключения плат расширения, их внешняя шина PCMCI может оказаться недостаточно быстрой для того, чтобы использовать скорость FireWire.

Рисунок (FireWire адаптор для шины PCI производства Maxtor)

Рисунок (Адаптор FireWire для шины PCMCI производства Synchrotech)

Провода и разъемы

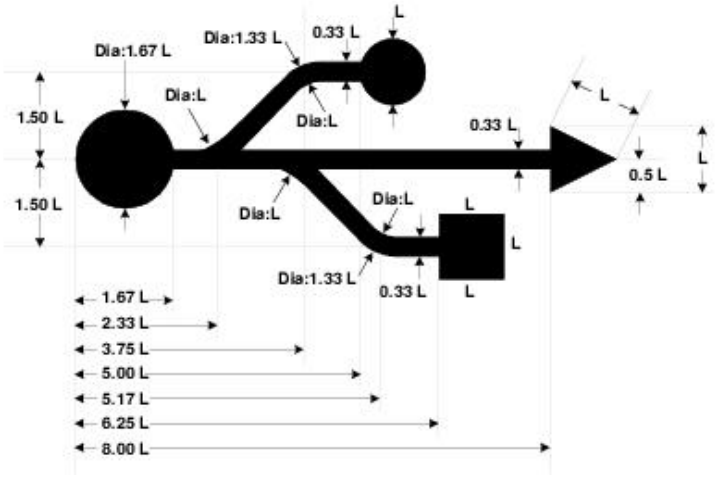
Устройства USB и FireWire предназначены в частности для того, чтобы при необходимости их можно было быстро и просто подключать и отключать друг от друга и от компьютера. При этом должен обеспечиваться хороший электрический контакт и устойчивость к износу от частого использования разъемов. Провода, соединяющие устройства могут быть достаточно длинными и находиться в области возможных электромагнитных помех от различных источников. Поэтому те и другие сконструированы с учетом этих требований.

Кабели и разъемы USB.

Разъемы типа “A” должны быть направлены от USB устройства к хосту (компьютеру). Конструктивно вилки и гнезда розеток имеют поперечные размеры 12 на 4.5 миллиметров и могут соединятся друг с другом единственным правильным способом. То есть неправильно вставить вилку в розетку невозможно. При соединении вилка заходит в розетку на глубину примерно 8 миллиметров. Розетки подобного типа мы можем найти с обратной стороны многих современных компьютеров. Примерами вилок типа “A” могут служить вилки, которыми снабжены USB клавиатуры и мыши. В случае клавиатур, мышей, псссивных HUB-ов не (не имеющих собственного источника питания), противоположный от вилки конец провода уходит внутрь самого устройства и не может быть отсоединен.

На USB разъемы или в непосредственной близи от них как правило наносится символика USB, определенная соответсвующей спецификацией.

Рисунок (Символика USB)



Эту символику можно видеть нанесенной на боковую поверхность большинства вилок.

Некоторые устройства (такие как сканеры, принтеры, активные HUB-ы) имеют отсоединяющийся USB кабель. Разъем который соединяет отсоединяющийся от устройства конец кабеля для удобства выполнен несовместимым с разъемами типа “A” и носит название

разъема типа “B”. Его поперечные размеры примерно 8.5 на 7.8 миллиметров. За счет скошенных углов на одной стороне этот разъем так же не может быть вставлен неправильно.

Существует еще один вариант Mini USB разема типа “B” с размерами около 6 на 2.7 миллиметров (который так же можно соединить только правильным способом). Данный тип разъемов предназначен для подсоединению к компьютеру миниатюрных устройств типа цифровых фотоаппаратов, мобильных телефонов, компьютеров налодонников. Таким образом на обоих концах таких кабелей находятся вилки разных типов “A” и “B” соответственно. Стоит обратить внимание что кабель не всегда входит в комплект поставки подобных устройств.

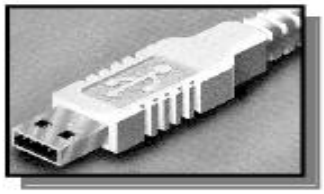
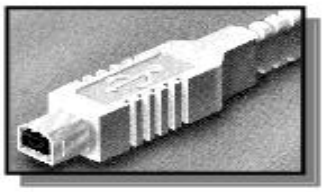
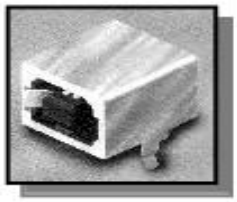
В отношении налодонников может естественным образом возникнуть желание не подключить к PC, а наоборот подключить к налодоннику другие USB устройства. В большинстве случаев – это невозможно так как с точки зрения шины USB налодонник сам является только лишь устройством но не хостом. Поэтому для такого подключения понадобится налодонник, который подобно PC имеет розетку типа “A”.

Кабели для USB устройств во избежании электромагнитных помех как правило являются экранированными. Использование неэкранированных проводов допускается стандартом только для низкоскоростных устройств (1.5Mbit/sec). Поэтому во избежании проблем со стабильностью работы для быстрых устройств (12Mbit/sec и более) настоятельно рекомендуется использовать экранированные кабели. К самим кабелям стандарт так же предъявляет ряд требований таких как время вносимой задержки сигнала (для кабеля в целом оно не должно превышать 30 наносекунд согласно спецификации USB1.0), максимальную величину ослабления сигнала (погонное сопротивление порядка десятых долей OM на метр), частотную зависимость ослабления сигнала (для USB1.0 используются частоты в диапазоне как минимум 16 мегагерц). Все эти цифры приводятся в этой главе для того, чтобы читатель мого составить общее впечатление об электрических характеристиках сигналов шины USB. За точной информацией следует обращаться например к главе 6 спецификации USB1.0

Кабели содержат 4 провода, два из которых предназначены для подачи по шине питающего напряжения от одного устройства к другому, два же других предназначены для передачи информационных сигналов в обе стороны. При этом для уменьшения влияния помех используется дифференциальная схема передачи сигналов.

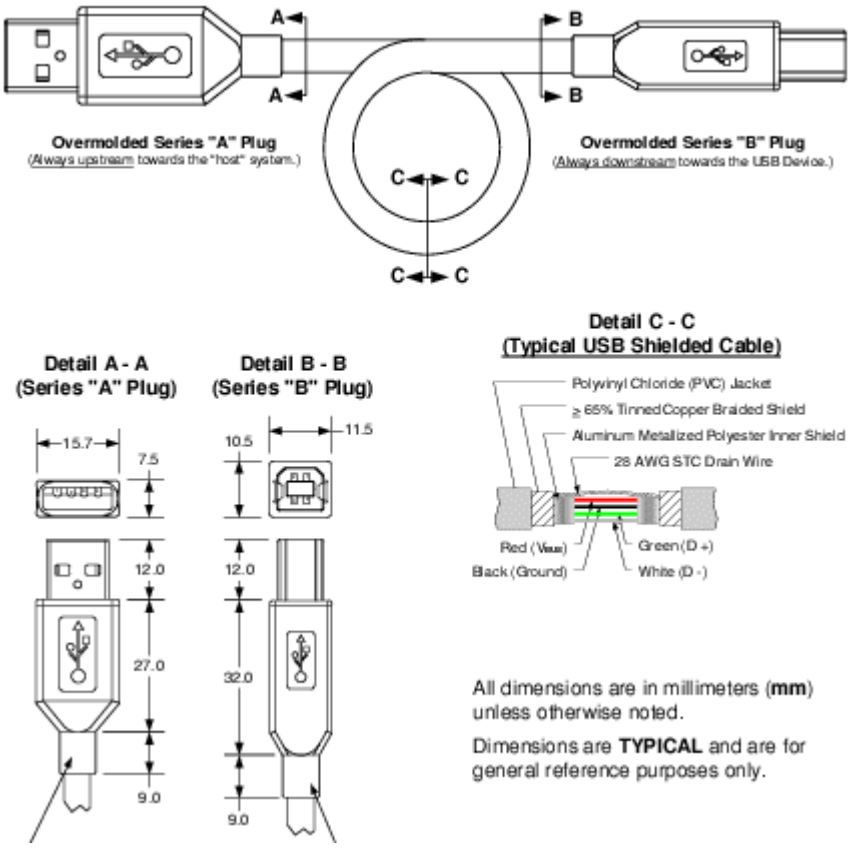
Провода, отвечающие за передачу сигналов, обычно выполнены в виде экранированной витой пары и имеют цвета зеленый (+Data) и белый (-Data). Питающее напряжение подается по красному проводу, земля имеет черный цвет.

Рисунок (коннекторы USB)

Series "A" Connectors	Series "B" Connectors
◆ Series "A" plugs are always oriented **upstream** towards the *Host System*	◆ Series "B" plugs are always oriented **downstream** towards the *USB Device*
 **"A" Plugs** (*From the USB Device*)	 **"B" Plugs** (*From the Host System*)
"A" Receptacles (*Downstream Output from the USB Host or Hub*) 	**"B" Receptacles** (*Upstream Input to the USB Device or Hub*) 

	Series "mini-B" Connectors
	◆ Series "mini-B" plugs are always oriented **downstream** towards the *USB Device*
	 **"mini-B" Plugs** (*From the Host System*)
	"mini-B" Receptacles (*Upstream Input to the USB Device or Hub*) 

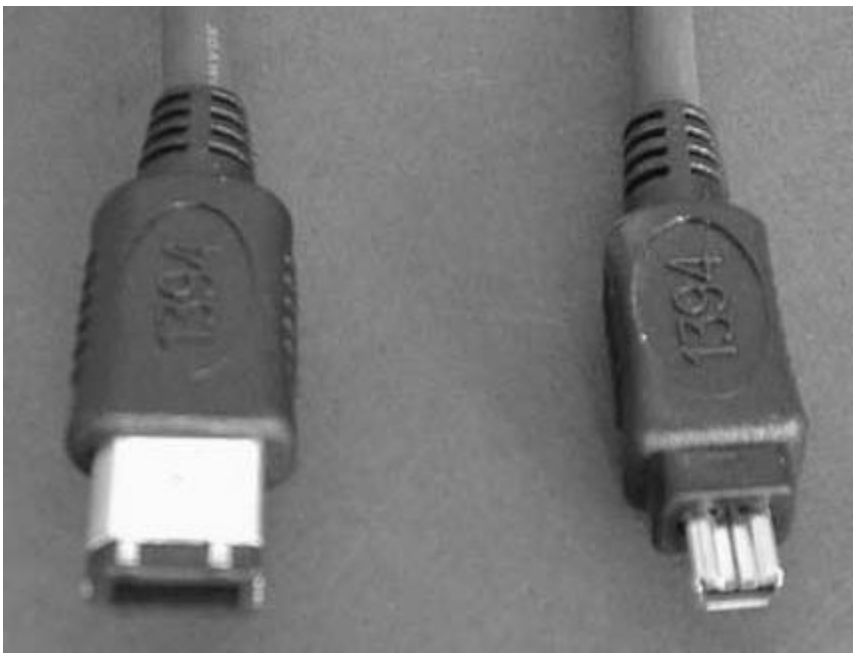
Рисунок (кабели USB)



Кабели и разъемы FireWire.

В отличие от USB разъемы и кабели FireWire могут как иметь провода для передачи питающего напряжения по шине так и не иметь их. В случае если провода для передачи питания имеются их общее количество составляет 6 (а не 4 как для USB). Аналогично, два провода предназначены для передачи питающего напряжения и четыре провода предназначены для передачи сигналов. Шестипроводные разъемы имеют размеры примерно 5.5 на 11 миллиметров, четырехпроводные примерно 5.5 на 3.5 миллиметров .

Рисунок (4 и 6 контактные разъемы FireWire)



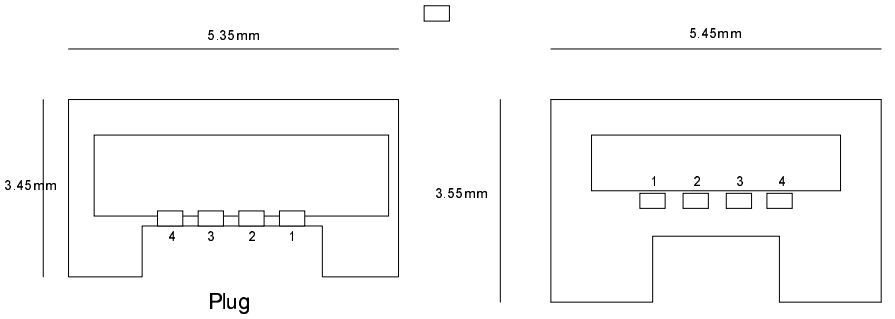
Четыре сигнальных провода объединены по два и носят названия TPA и TPB (Twisted pair A и B соответственно). При передаче устройства используют TPA для передачи дифференциального строба, а TPB для передачи дифференциального сигнала сигнала. При приеме сигнала – все наоборот. Сигнальные провода так же как и в случае USB выполнены в виде экранированной витой пары.

Кабели имеют на обоих концах разъемы типа “вилка”, а устройства разъемы типа “розетка”. При этом для соединения устройств имеющих розетки с разным числом проводов (4 и 6) используются кабели с соответствующими вилками, при этом 2 провода отвечающих за подачу питания на одном конце кабеля остаются незадействованными на другом. В нутри кабеля TPA и TPB меняются местами и приходят соответственно на другую пару разъемов вилки на противоположном конце. Цвета проводов следующие: земля – черный, питающее напряжение – белый, TPA – оранжевый и голубой, TPB – красный и зеленый. Разъемы устроены так, что при подсоединении их друг к другу контакты проводов, отвечающих за питание соединяются раньше чем контакты провода, отвечающих за передачу сигналов.

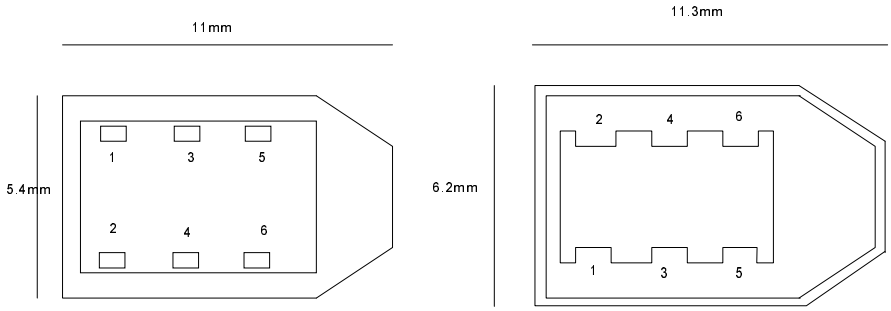
Так же как и в случае с USB вставить по ошибке вилку в розетку другой стороной невозможно.

Рисунок (кабель FireWire с двумя 6 контактными разъемами)

Рисунок (кабель FireWire с переходом с 6 контактного разъема на 4 контактный)

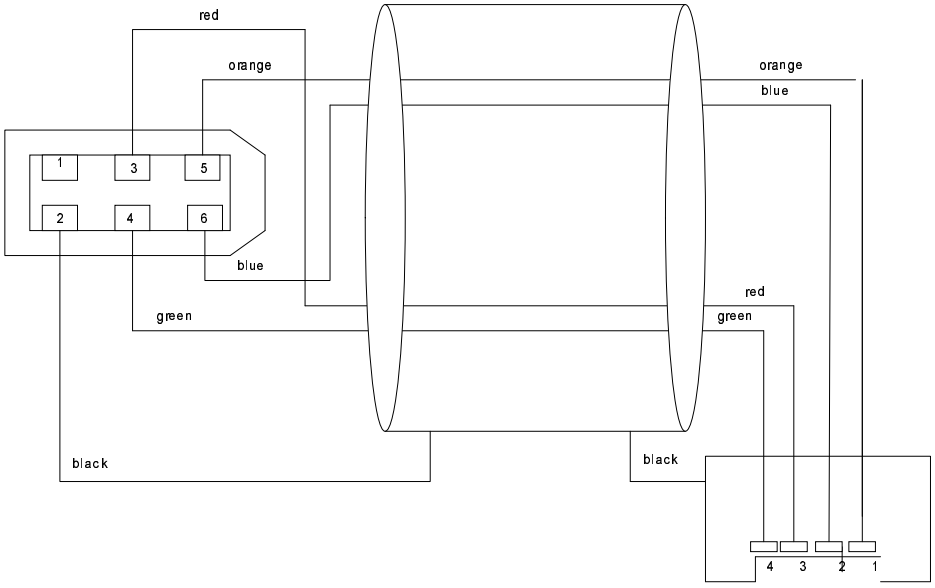


Plug



Plague

Socket



Модель передачи данных

Основные способы передачи данных (асинхронный и изохронный)

В зависимости от конкретного приложения для передачи данных при помощи последовательных шин USB и FireWire используются различные режимы передачи данных. Восновном можно выделить два режима асинхронный и изохронный (соответствующие англоязычные термины **asynchronous** и **isochronous**).

Первый режим используется когда необходима гарантированная доставка данных между двумя устройствами (речь идет о ситуациях когда одно устройство не обязательно является компьютером см.). Под гарантированной доставкой понимается то обстоятельство что в случае возникновения каких-либо ошибок (кроме случаев когда шина обнаружила что одно из устройств было физически отключено в процессе передачи) при передаче данных происходит попытка повторной передачи данных до тех пор пока данные не будут переданы успешно либо пока не будет исчерпан лимит количества повторных передач. Асинхронный режим может например использоваться для передачи данных между компьютером и внешним накопителем данных (CDR,CDRW,Compact Flash Disk и.т.д.). Понятно что во всех случаях с дисками на первом месте стоит сохранение целостности данных и только лишь во вторую очередь важно постоянство скорости их передачи. Как видно из данного выше описания синхронного режима передачи в процессе передачи могут иметь место задержки связанные с повторной посылкой пакетов данных. Следует так же принять во внимание конкуренцию разных устройств за канал передачи данных по шине, в силу этих обстоятельств к асинхронной передаче (как это и следует из ее названия) не предъявляется требований по величине временной задержки передачи данных между источником и приемником.

Совершенно иная ситуация имеет место при изохронном типе передаче, в этом случае во главу угла ставится именно достижение фиксированной величины задержки сигнала. Поэтому обе шины реализованы так что в случае ошибок при изохронной передаче повторной попытки передачи на уровне аппаратного обеспечения самой шины не происходит. Такой подход позволяет достичь минимального времени задержки в тех случаях когда целостность передаваемых данных не имеет особого значения (как например это иногда бывает при передаче телевизионного изображения, разумеется речь идет о ситуациях когда вероятность возникновения ошибок не высока, а например само изображение не отличается высоким качеством) при этом в случае необходимости коррекции ошибок это может быть достигнуто на более высоких уровнях протокола передачи чем сама аппаратная реализация шины. В последнем случае коррекцией ошибок может заниматься ПО уровня системных драйверов или прикладного уровня настольного компьютера или встроенного компьютера прибора участвующего в передаче данных по шине. Коррекция может аналогично происходить при помощи повторной передачи данных одним из устройств все через тот же используемый им логический изохронный канал передачи данных внутри шины. Разумеется такая коррекция требует обратной связи в реальном времени от устройства, являющегося приемником информации. Разумеется в случае коррекции за счет повторной передачи возрастает возможная задержка времени прихода данных от одного устройства к другому. Для обеспечения обратной связи может например использоваться еще один изохронный канал данных направленный в обратную сторону или еще одна разновидность передачи данных называемая в англоязычной литературе interrupt transfer или то же передача по прерыванию. Передача по прерыванию в полной мере относится к шине USB (в первоначальном варианте стандарта IEEE1394-1995 для шины FireWire аналогичное понятие отсутствует) и характеризуется тем что в течении некоторого периодически

заданного интервала времени гарантируется что одно устройство может послать другому пакет данных ограниченной длины а другое устройство гарантированно его примет в тот же интервал времени. Поэтому передача по прерыванию хорошо подходит именно для посылки сигналя обратной связи о отсутствии или наличии ошибок при изохронной передаче данных. Еще одним применением передачи по прерыванию может быть периодическое уведомление одного устройства о состоянии другого, например уведомления компьютера об окончании принтером печати очередной порции данных или аналогично уведомление модемом о наличии входящего звонка.

Передача по прерыванию в отличие от изохронной передачи по шине USB вместе с тем обеспечивает повторную передачу данных на уровне самой шины, при том что она наследует от самой изохронной передачи способность передавать данные с гарантированной максимальной задержкой. Передача по прерыванию предназначена для передачи пакетов от устройства к компьютеру. Однако с точки зрения программного управления потоком данных она гораздо ближе к асинхронной передаче чем к изохронной, поэтому неудивительно что в программах для получения компьютером данных через асинхронный канал и через канал передачи данных по прерыванию используются одни и те же функции за тем лишь исключением что в них передаются разные идентификаторы каналов.

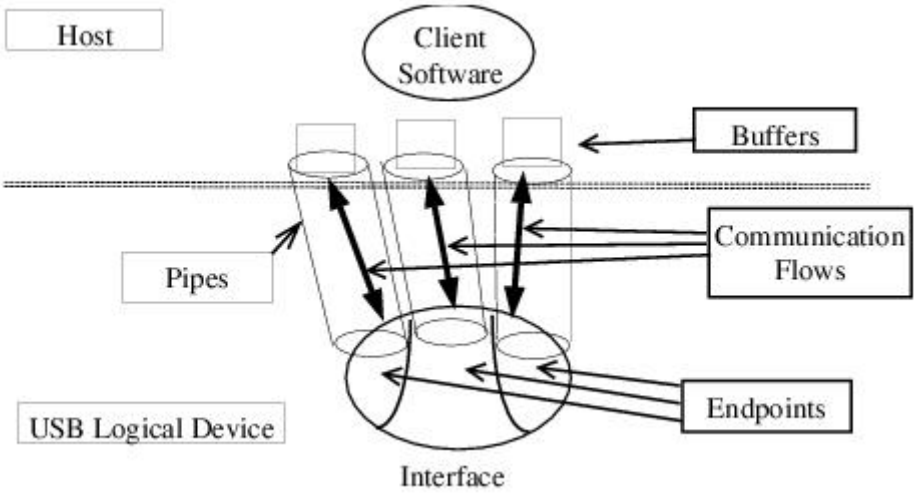
Обе описываемых здесь последовательных шины (USB и FireWire) для реализации описанного выше гибкого управления потоками данных при наличии множества одновременно подключенных устройств используют так называемую передачу данных при помощи тайм-слотов, состоящую в том что в течении периодически повторяющегося интервала времени (которое фактически и определяет достижимые самые короткие максимальные задержки) устройства подключенные к шине получают возможность передачи пакетов данных через используемые ими логические каналы. И здесь становится важным понятия арбитража на шине, определяющего порядок “справедливого” распределения времени между устройствами и соответственно логическими каналами в течении каждого тайм-слота. Для шин USB и FireWire в силу “генезиса их происхождения” этот процесс происходит существенно по разному (более подробное описание приводится в следующих разделах). Еще одним существенным отличием FireWire является наличие broadcast-режима изохронной передачи данных, когда несколько устройств могут одновременно получать изохронные данные, передаваемые одним устройством. Такой режим, в частности важен, например когда необходимо организовать одновременную запись одним устройством и воспроизведение другим видеоизображения, передаваемого в реальном времени цифровой видеокамерой

Существенным является отметить тот факт что в силу физических ограничений пропускной способности шины очевидно не всякий набор одновременно существующих логичеких каналов передачи может существовать внутри нее. Если какое-либо из устройств потребует для изохронного полосы пропускания (которая исчисляется в доле времени от тайм-слота используемого неким логическим каналом, связанным с этим устройством) что суммарная используемая всеми устройствами и их изохронными каналами полоса превысит значение отведенное на шине для изохронной передачи, такое устройство просто не сможет ее получить и просто останется не сконфигурированным. Использование этого устройства (или по крайней мере его изохронного канала) будет невозможно до тех пор пока другие устройства не освободят достаточную полосу для того, чтобы шина (совместно с ПО) могла это устройство сконфигурировать. Такое поведение шины является частью реализации механизма гарантированных временных задержек передачи данных.

В процессе конфигурирования устройств подключенных к шине так же используются асинхронные каналы передачи данных. Для FireWire и USB здесь существуют существенные отличия. В случае USB все взаимодействия между устройствами происходит посредством последовательных каналов передачи данных данных, соединяющих две логические единицы находящиеся в двух разных устройствах, подсоединенных к шине, и носящих (в соответствии со спецификацией USB) название **endpoint**. В отличие от этого, в

спецификации FireWire (IEEE1394) понятие канала используется только когда речь идет об изохронной передаче, во всех остальных случаях все подключенные к шине устройства (по-видимому несколько более архаично) представляются в виде виртуального адресного пространства в различные адреса которого может производиться запись и чтение.

Рисунок (Каналы endpoint-ы USB)



Для устройств, соединенных при помощи USB, каналы, отвечающие за конфигурирование устройств логически подключены к endpoint-ам носящим название control endpoint (управляющий endpoint), такие endpoint-ы в отличие от обычных асинхронных и изохронных осуществляют двунаправленную передачу данных между устройством и компьютером. Передача данных в такие endpoint-ы всегда происходит в соотвествии со специальным форматом, определяемым спецификацией USB. Каждое устройство имеет control endpoint по умолчанию, который всегда имеет номер 0 (уникальный идентификатор в пределах одного устройства).

FireWire для конфигурирования устройств использует в виртуальном адресном пространстве для каждого устройства CSR-структуру (Control state registers), определяемую спецификацией ISO/IEC-13213. В процессе конфигурирования производится запись в регистры, определенные этой спецификацией. Для каждого из устройств выделена огромная область памяти размером в 256 мегабайт в составе котрой находится его CSR. Стандарт IEEE1394 допускает объединение нескольких шин FireWire в одну, в этом случае частью адреса в виртуальном адресном пространстве становится еще номер шины.

Назначения способов передачи данных, устройства их использующие

В силу того, что FireWire изначально проектироваласть как шина для передачи видео и аудиопотоков между различными устройствами независимо от наличия компьютера, в настоящее время количество существующих устройств использующих изохронную передачу данных по этой существенно больше чем количество таковых для USB. К числу таких устройств относятся например видеокамеры производства Sony (некоторые из которых мы собираемся описать более подробно в следующих главах), некоторые устройства сбора данных например производства московской фирмы ЗАО “Инструментальные системы”. Последние могут быть интересны специалистам в области промышленных и научных измерений, а так же систем связи. С этими устройствами можно ознакомится на сайт фирмы www.insys.ru.

Изохронная передача по USB до сих пор считается “не тривиальным делом” на уровне ПО. И до сих пор USB устройства не всегда используют ее в тех случаях когда это было бы логично. Примером тому в частности могут служить столь распространенные WEB-камеры Logitech QuickCam, а так же некоторые CD/DVD RW производства Sony. Те и другие используют асинхронный режим передачи данных. В случае с WEB=камерами это вобщем-то оправданное упрощение конструкции в силу того, что WEB-камеры как правило работают в условиях довольно медленной связи по интернет и речь о каком-то realtime как правило не идет. В случае же с устройствами записи на оптичиские диски при некоторых учловияхиспользование асинхронной передачи может быть причиной того, что устройство не успеет вовремя получить от компьютера необходимый объем данных и оптический носитель информации (так называемая “болванка”) может быть безвозвратно испрочен. Решением проблемы в этом случае может быть программное переключение устройства в режим когда процесс записи будет осуществляться на более низкой скорости. Подобная неприятная ситуация может возникнуть как в случае чрезмерной загруженности шины USB другими устройствами, так и в случае когда host-контроллер USB компьютера является недостаточно быстрым так как соответствует более ранней спецификации USB1.0 или USB1.1.

В случае если бы устройство использовало изохронную передачу, то подобная ситуация не могла бы иметь место в силу того что шина просто не смогла бы сконфигурировать устройство для для работы со скоростью изохронной передачи необходимой для слишком высоких скоростей записи на оптичекий диск, таких что скорость подкачки по шине была бы недостаточной. Шина автоматически бы обнаружила нехватку полосы пропускания для передачи с высокой скоростью.

Положение с изохронной передачей вероятно изменится когда большее распространение получит скоростная шина USB2.0. В настоящее время многие выпускаемые компьютеры комплектуются host-контроллером USB2.0, однако число устройств, поддерживающих работу по USB2.0 невелико по сравнению с числом таковых для USB1.0.

Управление шиной

“Централизованное” управление шиной USB

Историю USB повидимому можно отсчитывать от ноября 1994, когда была создана версия 0.7 спецификации, но сегодня широкому кругу потребителей знакомы устройства совместимые со спецификациями USB1.0 и USB2.0, первая была принята в качестве промышленного стандарта в 1995 году.

Шина USB изначально была ориентирована на использование совместно с компьютером (она и задумывалась как замена Serial и LPT портам). Это означает что к шине подключен один единственный компьютер, который отвечает за все процессы управления шиной, а все устройства являются лишь “пассивными исполнителями его воли”. Поэтому следует различать интерфейы USB хоста и USB устройства. Поток управления всегда направлен от компьютера к одному или нескольким устройствам, подсоединенным к шине. И даже в том случае когда используется устройство которое якобы соединяет 2 компьютера по USB (подобно тому как это можно сделать при помощи последовательных портов) на самом деле с точки зрения каждого компьютера имеется единственное USB устройство в котором “неизвестно откуда” появляются данные, находящиеся в другом компьютере.

В задачу центрального узла USB шины (хоста) входит: определение топологии дерева подключенных устройств (в вершине дерева всегда находится хост), распределение адресов между обнаруженными устройствами, управление распределение электроэнергии (которая может подаваться к устройствам через шину), арбитраж шины, выработка тайм-слотов и переодичексий опрос устройств, управление ресурсом пропускной способности шины (полосы) и распределение ее между устройствами и.т.д.

USB устройства могут иметь собственный источники питания вне шины, в этом случае согласно пункту 7.2.1 спецификации USB версии 1.0 (принятой в начале 1996 года) такие устройства не должны пытаться передать часть энергии в шину для питания других устройств. Это как раз связано с тем, что управление питанием USB строго централизовано и может обеспечивать режимы полного или частичного выключения питания отдельных ветвей дерева USB устройств. Частичное исключение составляют HUB-ы, имеющие собственный источник питания, они могут подавать питающее напряжение вних по шине в сторону противоположную от хоста. Тем не менее они тоже должны строго следовать командам, получаемым от хоста и отвечающим за управление питанием.

После включения компьютера его хост-устройство последовательно обнаруживает все USB устройства, составляет топологическую схему дерева, сообщает устройствам их адреса, и начинает в течении интервала времени тайм-слота опрашивать все активные устройства.

Характерно то, что благодаря асимметричности разъемов USB кабелей, практически невозможно подключить устройства так, что нарушится однонаправленная структура дерева устройств (пункт 6.1 спецификации USB1.0). Благодаря тому, что HUB-ы должны иметь неотключаемый кабель с коннектором вилка типа “A” на конце, невозможно создать “закольцованное” дерево. Однако если найдется комбинация устройств (например с использованием устройства для USB связи между двумя компьютерами и HUB-а) то такое закольцовывание окажется возможно. Шина USB не приспособлена для работы в таких конфигурациях, и естественно возникнут проблемы в ее работе, впрочем на практике вряд ли кому-либо придет в голову собрать подобное.

Тем не менее сходная проблема вполне жизненна в отношении шины FireWire (в силу симметричности разъемов на кабелях), и мы собираемся здесь ее рассмотьреть.

“Самоорганизующееся” управление шиной FireWire (IEEE1394)

В отличие от USB, шина FireWire проектировалась как шина для объединения потенциально равноправных устройств ни одно из которых изначально не является выдкленным устройством – компьютером с большим количеством памяти, высоким быстродействием способным взять на себя все функции по управлению ниной (хотя в настоящее время к FireWire можно подключить несколько компьютеров одновременно). Шина должна была обеспечивать высокие скорости, достаточные для передачи по ней аудио и видео потоков данных. Именно эти ее свойства и стали причиной того, что FireWire получила столь широкое распространение в мире любительских и профессиональных видеокамер Sony. А способность транслировать изображение на несколько принимающих устройств одновременно, не занимая для каждого из них дополнительной полосы пропускания сделали FireWire поистине уникальным решение для передаче телевизионных и других видеосигналов между устройствами цифровой видеостудии.

Изначально работы по сосданию FireWire велись в начале 1980 годов фирмой Apple Computer Inc., однако другие производители тоже заинтересовались возможностью применения этой шины для их устройств. В итоге был создан специальный комитет по FireWire и в 1995 году увидела свет первая версия стандарта IEEE1394-1995. Более поздние версии IEEE1394 обратносовместимы с этой версией и IEEE1394-1995 в настоящее время используется во многих устройствах FireWire

Повидимому в частности из за более сложной структуры (без выделенного центрального компьютера) реализация FireWire оказывается более дорогоий чем USB. Для большинства компьютерных переферийных устройств середины-конца 1990 годов была характерна относительно небольшая скорость передачи данных. Количество таких устройств, одновременно подключенных к компьютеру так же было невелико. Существовавшая в уже те времена спецификация Serial Plug&Play позволяла ОС Windows достаточно легко определять тип подключенных к последовательным портам устройств, для подключения большого количества таких устройств с давних времен были доступны мультипортовые платы. В тех же случаях когда действительно требовалось “на ходу” подключать к компьютеру устройства и обеспечивать работу на повышенных скоростях USB1.0 прекрастно справлялась с такой задачей. Таким образом долгое время FireWire оставаясь “игрушкой” для любителей видеокамер Sony.

В силу всего вышесказанного широкое распространение в компьютерной области FireWire стала приобретать лишь только с появлением на рынке большого количества уомпъютерных мультимедийных устройств, требующих высокой скорости передачи данных. И сегодня мы можем найти в магазинах относительно недорогие платы host-контроллеров FireWire для IBM-совместимых персональных компьютеров, а так же для ряда альтернативных архитектур таких как Mac, Power PC и.т.д. В некоторые компьютеры типа Notebook шина FireWire уже интегрирована и соответствующий разъем как правило бывает выведен на заднюю панель.

С целью продвижения на рынок аппаратных решений для FireWire, разработчики стандартов создали спецификацию электронного интерфейса чипа контроллера FireWire, которая получила название OHCI (Open Host Controller Interface). Данная спецификация является открытой и может быть найдена в Интернете. Существует так же ряд коммерчески доступных чипов, реализующих интерфейс OHCI. В настоящее время ряд отечественных производителей электронного оборудования (например Московская фирма ЗАО “Инструментальные Системы”) используют в ряде серийно выпускаемых ими устройств OHCI чипы и шину FireWire.

Рассмотрим процесс управления шиной FireWire более подробно. В отличие от USB шина IEEE1394 не имеет заранее выделенного управляющего узла и функции такового после того как шина окажется сконфигурированной могут быть распределены между несколькими устройствами на шине. Важно отметить то, что в случае отсутствия устройств,

способных выполнять некоторый функции сама шина останется работоспособной, но эти функции останутся недоступными. Так же важен то факт что любое подключение или отключение устройства способно разрешить или наоборот запретить выполнение той или иной функции каким-либо из подключенных устройств. Поэтому шина должна автоматически перераспределить функции управления ей самой между имеющимися устройствами. Все эти действия происходят в процессе самоинициализации шины.

В общих чертах этот процесс состоит состоит из следующих этапов:

1. Bus initialization
2. Tree identification
3. Self identifivcation

Во время **Bus initialization** все устройства получают уведомления о том, что происходит процесс инициализации шины. При этом каждое устройство должно перейти в состояние **idle state**, шина ожидает в течении некоторого заранне предопределенного времени чтобы все устройства успели перейти в это состояние. В связи с тем что к шине могут быть подключены устройства, поддерживающие различные максимальные скорости весь процесс конфигурации использует только минимальную скорость передачи 100Mb/sec.

Tree identification состоит в том, автоматически выбирается устройство, которое будет выполнять роль корневого узла (**root node**). Подробнее этот процесс описан в следующих разделах. В картце его можно описать следующим образом: сначала устройства, которые имеют лишь только одно соединение с соседним устройством помечают это соединение как направленное в сторону корневого узла, а соответствующие им устройства ставшие их родительскими помечают соединение как “уже определенное”. После чего оказывается (если собрана допустимая схема взаимного подключения устройств) что хотя бы у одного из найденных родительских узлов всего лишь одно соединение является “еще не определенным”, тогда оно так же как дочерние узлы самого нижнего уровня на первом шаге помечает его как направленное в сторону корневого узла и весь процесс повторяется. В конце концов оказывается один или несколько устройств “кандидатов” на роль корневого узла. Из этих узлов выбирается один, который и станет корневым узлом шины FireWire. Понятно что если топология подключения устройств содержит кольца, то шина не сможет определить корневой узел и соответственно не будет работать.

Последний этап **Self identifivcation** состоит в том, что корневой узел, производит последовательны обход всех ветвей дерева и назначает каждому узлу (устройству) свой адрес, по которому будет происходить обращение к нему в процессе работы шины.

После завершения этих этапов из имеющихся устройств (в зависимости от наличия подключенных устройств, подходящих для выполнения той или иной функции) выбираются устройства, отвечающие за ту или иную часть управления шиной, а именно:

1 .**Cycle Master –** устройство отвечающее за начало изохронных транзакций через равные 125 миллисекундные интервалы времени.

2. **Isochronous Resource Manager –** устройство, которое отвечает за выделение по требованию других узлов изохронных каналов передачи с требуемой шириной полосы.

3. **Bus Manager –** устройство, отвечающее за поддержание актуальной схемы текущей топологии шины (для предоставления ее другим устройствам в случае необходимости), управляющее подачей питания к устройствам на шине, управляет резервированием полосы для асинхронной передачи.

Совершенно не обязательно что корневой узел будет сочетать в себе функции всех этих трех узлов.

Скорость передачи необходимая для работы различных устройств может быть различна (100Mb/sec,200Mb/sec,400Mb/sec и.т.д.) поэтому для достижения максимальной производительности при подключении устройств следует следить за тем, чтобы на пути передачи данных между двумя устройствами все промежуточные устройства поддерживали скорость передачи не меньше чем требуется для связи выбранных двух устройств. Понятно

так же что если речь идет об изохронной передаче, то в случае недостаточной пропускной способности промежуточных узлов такая передача работать не будет.

Управление питанием FireWire существенно сложнее чем в USB, устройства могут отдавать часть энергии своего внутреннего источника питания в шину для питания других устройств. При этом так как некоторые кабели могут не иметь проводов для передачи питающего напряжения – вся сеть устройств, подключенных к шине может распасться на несколько доменов с раздельным питанием.

Определение топологии шины

USB

Передача любых пакетов в шине USB происходит только “по команде сверху” (устройство, получившее Token-пакет от хоста может передать данный в шину), поэтому даже для обнаружения таких асинхронных событий как подсоединение и отсоединение устройств и используется опрос центральным узлом (хостом, роль которого обычно выполняет PC) всех HUB-устройств имеющихся на шине. Для единообразия считается что хост содержит так называемый корневой HUB (root HUB), управление которым происходит примерно так же как и HUB-устройствами, подключенными снаружи.

Рисунок (Пример фзической топологии)

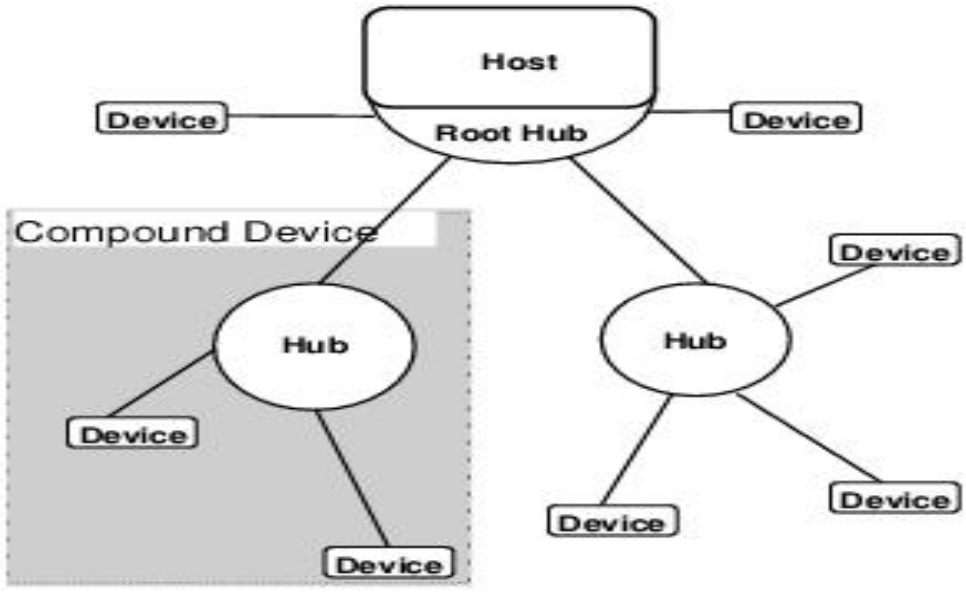
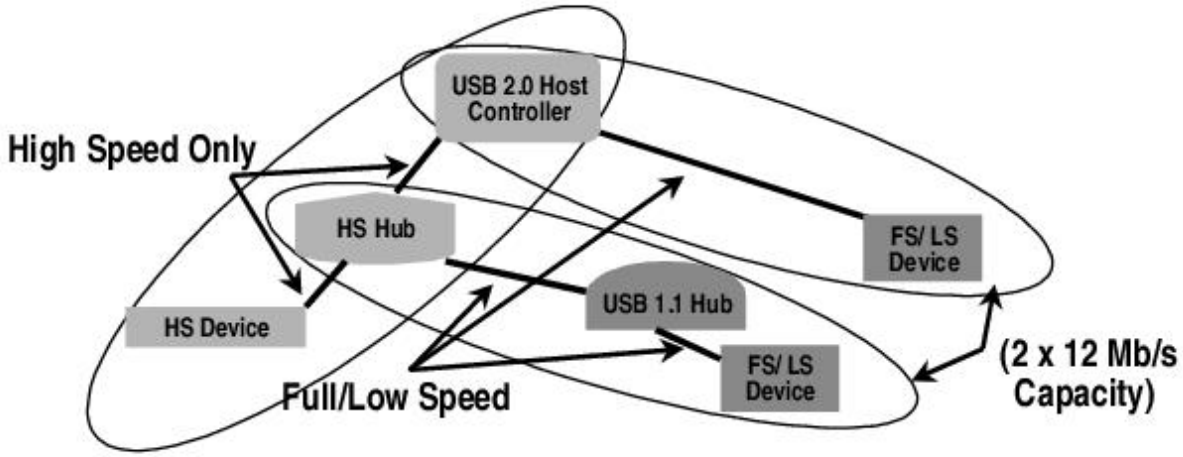


Figure 5-5. USB Physical Bus Topology

Отличием корневого HUB, может быть в частности то, что он может логически объединять в себе корневые узлы нескольких шин USB (соответственно с максимально возможным числом подключаемых к каждой шине устройств равным 127). Для каждой такой шины назначение 7-битных адресов устройствам происходит раздельно. Однако с точки зперния прикладного ПО (которое редко интересуется точным значением физического адреса конкретного устройства) все это вместе может выглядеть как “одна большая USB

шина”. Интересно отметить что, объединенных таким образом шины нередко даже поддерживают разные версии стандарта (USB1.0 и USB2.0), что не мешает прикладным программам и драйверам при необходимости воспринимать их как единое целое.

Рисунок (Топология, включающая USB1.0 и USB2.0 устройства)



Каждому вновь подключенному устройствухост назначает уникальный в рамках одной физической шины 7-битный адрес, в дальнейшем все пакеты, направляемые этому устройству будут идентифицироваться именно этим адресом.

Рассмотрим этот процесс назначения адресов подробнее:

Обращение к каждому устройству происходит путем отправки и приема данных между хостом и логическими концевыми точками устройств, называемыми endpoint. Устройства могут содержать в себе несколько endpoint-ов. С каждым из них хост может установить логический канал связи pipe.

Каждый endpoint аналогичен телефонному аппарату, а каждый pipe телефонному проводу, с той лишь разницей, что каналы pipe не обязательно могут быть двунаправленными.

Сейчас для нас важно то, что каждое устройство всегда имеет двунаправленный канал с номером 0 (default pipe 0) и что именно он используется хостом в процессе конфигурирования устройств, в частности для назначения им адресов. В отличие от FireWire устройства не могут обмениваться друг с другом информацией напрямую, все логические каналы могут соединять устройства лишь с хостом. Поэтому процедура управления шиной для USB в целом выглядит существенно проще чем для IEEE1394.

Для обмена данными с управляющим каналом USB устройства (default pipe 0), хост использует специальную разновидность асинхронной передачи данных, называемую control transfer. Для нее характерно наличие строго определенного формата передаваемых в обе стороны данных. Для обнаружения подключения и отключения устройств к HUB-ам, образующим разветвления в топологии шины используется передача данных по прерыванияю (interrupt transfer), которая кратко обсуждалась во вводной части.

Каждое устройство (в том числе и HUB), которое хочет использовать interrupt transfer в процессе его конфигурирования хостом заявляет о наличии у него концевой точки типа interrupt endpoint и сообщает хосту как часто необходимо опрашивать. Таким образом каждое хост периодически опрашивает каждое HUB-устройство и если соответствующий interrupt endpoint возвращает через свой логический канал связи (interrupt pipe) сообщение о том, что произошли изменения в числе подключенных к данному HUB устройств, хост

автоматически считывает через default pipe 0 из внутренних регистров HUB информацию о том, что же именно произошло.

Если произошло подключение нового устройства, то следующим шагом является назначение адреса этого нового устройства. Вновь подключенное устройство поумолчанию имеет адрес 0, поэтому хост должен просто послать новому устройству через его default pipe 0 команду назначения адреса!

В случае отключения устройства хост помечает, использовавшийся этим устройствам, адрес как свободный и сообщает компонентам программного обеспечения, использовавшим данное устройство о его физическом отключении.

Поскольку к шине могут подключаться устройства, поддерживающие различные скорости передачи данных, следует обсудить подробнее, то как это отражается на процессе инициализации шины и назначения адресов.

В настоящее время устройства могут быть разделены на следующие типы: определенные спецификацией USB1.0 устройства low speed devices (1.5Mbit/sec) и full speed devices (12Mbit/sec), а так же USB2.0 устройства hight speed devices (480Mbit/sec). При чем HUB-ы всегда работают со скоростью full или high.

Первые два типа устройств различаются на уровне электрического протокола, в частности low speed устройства имеют резистор положительного смещения подключенным к сигнальному проводу “D-”, в свою очередь full speed устройства имеют таковой подключенный к “D+”. Что же касается hight speed устройств, в целях достижения аппаратной совместимости USB1.0 и USB2.0 в процессе инициализации шины они выглядит как full speed устройства. Однако в процессе конфигурирования такие устройства сообщают хосту, что способны работать на скорости 480Mbit/sec и впоследствии хост может использовать при обращении к ним именно эту скорость.

Для исключения возможности неправильной интерпретации более медленными устройствами пакетов, предназначенных для более быстрых устройств используется “принцип разделения по скоростям”, состоящий в том, HUB-устройства, зная с какими максимальными скоростями могут работать устройства подключенный к их портам, передают пакеты относящиеся к той или иной скорости передачи только тем устройствам, для которых данная скорость является допустимой.

HUB-устройства, способные работать на скорости 480Mbit/sec используют ее для обмена пакетами с хостом, в то время как для подключенных к ним low/full speed устройств данные специально транслируются HUB-ом на болнн низкой скорости.

Для двух основный версий USB на электрическом уровне так же различается допустимая длительность таймслотов: длительность фрейма USB1.0 всегда составляет 1 миллисекунду, более быстрые USB2.0 устройства используют 125 микросекундный микрофрейм. Микрофреймы передаются только high speed устройствам.

Если какое-либо быстрое устройство окажется подключенным к хосту через медленный HUB, очевидно что оно как минимум не сможет использовать возможности передачи данных с высокой скоростью, этот факт следует принимать во внимание при использовании HUB-ов.

Благодаря тому, что в процессе инициализации шины устройства сообщают хосту уникальные для каждой модели устройств параметры VendorID и ProductID (характеризующие соответственно производителя и саму модель устройство), ОС хоста имеет возможность произвести автоматический поиск драйвера, поддерживающего работу с данным типом устройства.

Ниже приводятся таблицы, описывающие структуры запросов передаваемых хостом через default pipe 0 а так же получаемых в ответ от устройств дескрипторов, общих для всех устройств, а так же специфичных для HUB-устройств. Полностью все таблицы можно найти в Интернет в PDF-файлах, содержащих спецификации USB1.0 и USB2.0 (главы 9-11), мы приводим основные таблицы для полноты изложения.

Таблица (формат заголовка пакета для control transfer, передаваемогог через default endpoint)

Смещение (байт)	*Поле*	*Размер*	*Тип*	*Описание*
0	bmRequestType	1	Битовая маска	Бит 7 – направление передачи (1 – от хоста к устройству, 0 – наоборот) Биты 6 и 5 – тип запроса: 00 – standars 01 - device class 10 – vendor specific 11 – зарезервировано Биты 4,3,2,1,0 получатель пакета внутри устройства: 00000 - device 00001 - interface 00010 - endpoint 00011 - другие остальные значения зарезервированы (биты отсчитываются с права на лево)

Смещение (байт)	*Поле*	*Размер*	*Тип*	*Описание*
1	bRequest	1	Число	Тип запроса (расшифровка стандартных значений в след. таблице)
2	wValue	2	Число	Дополнительный параметр к запросу
4	wIndex	2	Число	Дополнительный параметр, используемый как индекс или смещение
6	wLength	2	Число	Длина передаваемых данных

Таблица (Стандартные типы запросов)

bmRequestType	*bRequest*	*wValue*	*wIndex*	*wLength*	*Data*
00000000B 00000001B 00000010B	CLEAR_FEATURE	Feature Selector	Zero interface endpoint	Zero	None
10000000B	GET_CONFIGURATION	Zero	Zero	One	Configuration value
10000000B	GET_DESCRIPTOR	Descriptor type and index	Zero or language ID	Descriptor lngth	Descriptor
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate interface
10000000B 10000001B 10000010B	GET_STATUS	Zero	Zero interface endpoint	Two	Device, interface or endpoint status
00000000B	SET_ADDRESS	Device address	Zero	Zero	None
00000000B	SET_CONFIGURATION	Configuration value	Zero	Zero	None
00000000B	SET_DESCRIPTOR	Descriptor type and index	Zero or Language ID	Descritor Length	Descriptor
00000000B 00000001B 00000010B	SET_FEATURE	Feature selector	Zero interface endpoint	Zero	None
00000001B	SET_INTERFACE	Alternate setting	Interface	Zero	None
10000001B	SYNCH_FRAME	Zero	Endpoint	Two	Frame number

Таблица (значения bRequest)

Brequest	*Value*
GET_STATUS	0
CLEAR_FEATURE	1
зарезервировано	2
SET_FEATURE	3
зарезервировано	4
SET_ADDRESS	5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNCH_FRAME	12

Таблица (типы дескрипторов)

Descriptor Type	*Value*
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4

Таблица (дескрипторы устройств, возвращаемые при помощи control transfer)

Смещение (байт)	*Поле*	*Длина (байт)*	*Тип*	*Описание*
0	bLength	1	число	Длина всего дескриптора в байтах
1	bDescriptorType	1	константа	DEVICE descriptor type
2	bcdUSB	2	BCD (binary coded decimal)	Версия поддерживаемого стандарта USB (например для версии 2.10 значение будет 0x210)

Смещение (байт)	*Поле*	*Длина (байт)*	*Тип*	*Описание*
4	bDeviceClass	1	Class	Класс устройства, например CDROM, принтер и.т.п. (подробнее см. на www.usb.org). Если значение равно 0, то каждый интерфейс в устройстве имеет собственный класс и все интерфейсы могут функционировать независимо друг от друга. Значения от 0x01 до 0xFE, учтройство поддерживает различные классы для разных интерфейсов, но их раздельное функционирование невозможно. Значение 0xFF отведено для vendor specific класса.
5	bDeviceSubClass	1	SubClass	Дополнительный параметр к классу устройства, если bDeviceClass установлен в 0, то так же должен иметь значение 0, для всех остальных значений bDeviceClass кроме 0xFF зарезервирован для назначения USB.
6	bDeviceProtocol	1	Protocol	Указывает поддерживается ли протокол, специфичный для класса устройств, значение 0 говорит о том, что поддерживает класс-специфичные протоколы не поддерживаются для устройства в целом, но могут поддерживаться для отдельных интерфейсов.
7	bMaxPacketSize0	1	число	Максимальная длина control пакетов для endpoint 0, допустимые значения: 8,16,32,64.
8	idVendor	2	ID	VendorID
10	idProduct	2	ID	ProductID
12	bcdDevice	2	BCD	Версия реализации устройства
14	iManufacturer	1	индекс	Индекс строкового дескриптора, содержащего название производителя
15	iProduct	1	индекс	Индекс строкового дескриптора, содержащего название устройства как продукта
16	iSerialNumber	1	индекс	Индекс строкового дескриптора, содержащего серийный номер устройства
17	bNumConfigurations	1	число	Количество возможных конфигураций, поддерживаемых устройством

Таблица (дескриптор configuration)

Смещение (байт)	*Поле*	*Длина (байт)*	*Тип*	*Описание*
0	bLength	1	число	Размер дескриптора в байтах
1	bDescriptorType	1	константа	CONFIGURATION
2	wTotalLength	2	число	Общая длина возвращаемых для этой конфигурации данных (включая саму конфигурацию и таковые для interface, endpoint, а так же специфичные для Vendor и Class)
4	wNumberInterfaces	1	число	Число поддерживаемых конфигурацией интерфейсов (interface)
5	bConfigurationValue	1	число	Значение, которое следует использовать для выбора данной конфигурации при помощи запроса SET_CONFIGURATION
6	iConfiguration	1	индекс	Смещение для строкового дескриптора, описывающего данную конфигурацию
7	bmAttributes	1	битовая маска	Битовые флажки, которые могут быть указаны в различных комбинациях (в частности биты 6 и 7 могцт быть установлены одновременно), указывают источник питания, используемый конфигурацией бит 7 – Bus powered бит 6 – Self powered бит 5 – Remote wakeup supported биты 4...0 – зарезервированы (равны 0) (биты отсчитываются с права н алево) Используемый в данный момент источник можо узнать при помощи запроса GET_STATUS
8	MaxPower	1	Значение в миллиампер умножить на два	Максимальный потребляемый ток (значение 50 означает 100 миллиампер)

Таблица (дескриптор interface)

Смещение (байт)	*Поле*	*Длина (байт)*	*Тип*	*Описание*
0	bLength	1	число	Размер дескриптора в байтах
1	BdescriptorType	1	константа	Тип дескриптора INTERFACE
2	bInterfaceNumber	1	число	Номер интерфейса (начиная с 0)
3	bAlternateSetting	1	число	Значение для выбора альтернативных установок для данного интерфейса
4	bNumEndpoints	1	число	Число endpoint-ов для данного интерфейса, не включая default endpoint
5	bInterfaceClass	1	Class	Класс устройства (см. www.usb.org), 0x00 означает что устройство не попадает ни в один стандартный класс, 0xFF означает что класс vendor specific
6	bInterfaceSubClass	1	SubClass	Подкласс, используется совметно с bInterfaceClass. Должен быть установлен в 0 если последний установлен в 0
7	bInterfaceProtocol	1	Protocol	Код поддерживаемого класс-специфичного протокола или 0 если ни один такой протокол не поддерживается. Может принимать значение 0xFF если для данного класса поддерживается vendor specific протокол
8	iInterface	1	Индекс	Смещение строкового дескриптора с описанием для данного интерфейса

Таблица (дескриптор endpoint)

Смещение (байт)	*Поле*	*Длина (байт)*	*Тип*	*Описание*
0	bLength	1	число	Размер дескриптора в байтах
1	bDescriptorType	1	константа	Тип ENDPOINT
2	bEndpointAddress	1	Endpoint (битовая структура)	Адрем endpoint-а. Биты 0-3 собственно номер, биты 4-6 зарезервированы, бит 7 направление (для control не имеет значения) для остальных 0 означает OUT, а 1 означает IN (IN,OUT с точки зрения хоста)

Смещение (байт)	*Поле*	*Длина (байт)*	*Тип*	*Описание*
3	BmAttributes	1	Битовая маска	Определяет тип endpoint-а в состоянии когда он сконфигурирован при помощи bConfigurationValue. Биты 0 и 1: 00 - Control 01 - Isochron 10 - Bulk 11 – Interrupt
4	wMaxPacketSize	2	число	Максимальный размер пакета, допустимый для данного endpoint-а. Эта величина, в частности, используется хостом при выделении полосы пропускания для изохронных каналов, фактичекси же изохронные каналы могут использовать и меньшую полосу.
6	bInterval	1	число	Период времени в миллисекундах как часто следует хосту запрашивать данный endpoint на предмет наличия в нем данных для передачи. Имеет значение только для interrupt (значения от 1 до 255), для изохронных endpoint-ов согласно USB1.0 должен быть установлен в 1, что соотвествует длительности таймслота, определенной версией стандарта 1.0

Структура строкового дескриптора, на который ссылаются приведенные выше таблицы описывается в таблице следующей:

Таблица (строковый дескриптор)

Смещение (байт)	*Поле*	*Длина (байт)*	*Тип*	*Описание*
0	bLength	1	Число	Размер дескриптора в байтах
1	bDescriptorType	1	константа	Тип STRING

Смещение (байт)	*Поле*	*Длина (байт)*	*Тип*	*Описание*
2	bString	N	Массив	Строка в представлении UNCODE

FireWire

Tree Identification.

При передаче сигналов по шине IEEE1394 все устройства используют механизм арбитража, реализованный на аппаратном уровне. Таким образом достигается разделение между ними проводного канала связи по времени. Сигнал строба посылается передающим устройством через пару проводов TPA а получается принимающим через TPB (напомним что пары проводов меняются местами внутри кабеля), аналогично данные передаются через TPB и принимаются через TPA. Сигналы так же участвуют в процессе арбитража, о котором более подробно рассказывается в главе о передаче сигналов.

Tree Identification использует два арбитражных сигнала:

1. Parent notify
2. Child notify

Сигнал Parent notify подается путем установки TPA в состояние “0” а TPB в состояние с высоким импедансом “Z” (так же называемое idle state). В отличие от этого Child notify устанавливает TPA в состояние “1”.

Parent notify – служит для подачи узлом шины (устройством) соседнему узлу о том что он намерен считать соседний узел своим родительским. Соответственно узел подает Child notify соседнему если он ранее получил от него сигнал Parent notify и в результате переговоров с другими узлами у него нет сомнений что соседний узел следует считать своим дочерним узлом.

Рассмотрим подробнее всю процедуру, приводящую к инициализации топологии шины, под “проинициализированной топологией” понимается такое состояние шины, когда:

1. Шина может быть представлена (возможно не единственным способом) как дерево с единственной вершиной
2. Из всех возможных вариантов один узел выбран в качестве вершины (Root node), узел выбранный в качестве вершины знает о своей роли в дереве
3. Каждый вершина знает к какому из его портов подключен его родительский узел (узел расположенный ближе к вершине дерева), к каким портам подключены дочерние узлы и какие порты свободны.

Каждый узел, который обнаруживет только один единственный подключенный к нему соседний узел подает ему сигнал Parent notify и начинает ожидать от него подтверждения в виде сигнала Child notify. Поскольку используется аппартный арбитраж, то если 2 узла собрались одновременно послать друг другу сигнал, один из них “захватит” шину раньше другого и соответственно его опередит. Если какой-либо узел на шине обнаружит что все его соединенные с соседями порты кроме одного получили сигнал Parent notify, то этот узел может послать на эти порты подтверждение Child notify и устройства на этих портах станут его дочерними узлами!

Узлы у которых остался единственный не идентифицированный таким способом подключенный порт, могут попытаться послать через него соседнему устройству сигнал Parent notify. В тот момент когда один из таких сигнал захочет послать Parent notify возможно соседний узел захочет послать такой же сигнал навстречу. Но опять один из двух узлов окажется быстрее, подобно тому как это случается с героями вестернов. Аналогию с кино можно продлить еще дальше: когда два узла пытаются одновременно предъявить друг другу сигнал Parent notify, они пытаются установить уровни “0” для TPA и “Z” для TPB (вспомним еще раз что пары меняются в кабеле местами), в результате входные компараторы обоих узлов увидят сигнал “0” как TPA на так и на TPB. Обнаружив это условие (называемое ROOT_CONTEND) оба узла могут случайным образом выбрать подождать ли им короткое время (около 0.25 микросекунд) или подождать чуть подольше (порядка 0.6 микросекунд)

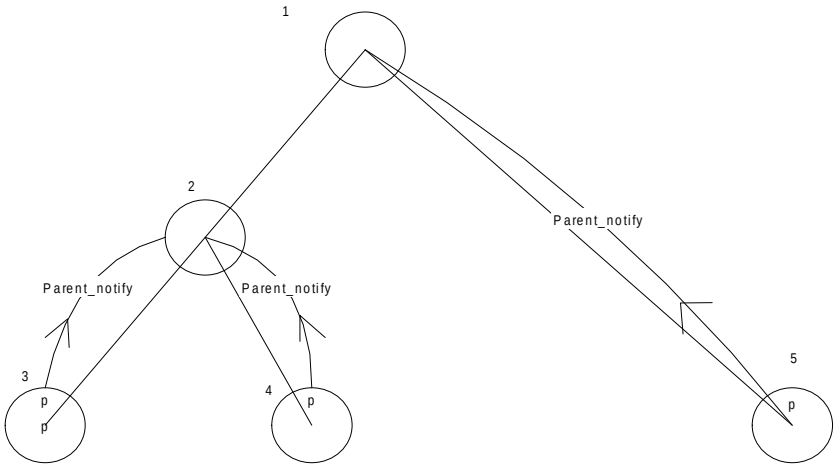
перед тем как повторить попытку. Как тут не вспомнить Голливудское: “У тебя слишком большая фора, Билл!”. При следующей попытке условие может повториться, однако после нескольких ходов своеобразной “Русской рулетки” всеже определится победитель.

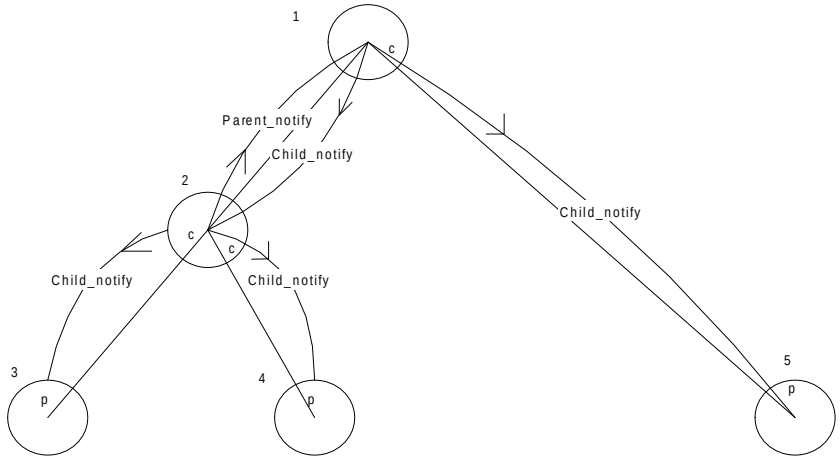
Вообще следует заметить что в силу того, что шина FireWire может не иметь выделенного управляющего центра, каковым для USB является компьютер, порядок в котором в ней происходят е процессы передачи управляются сигналами арбитража и таймаутами. Часто именно длительность используемых различными узлами таймаутов определяет основные характеристики происходящих процессов (в частности разделения полосы пропускания между различными типами передачи – асинхронными и изохронными). Использование таймаутов для определения приоритетов передачи данных знакомо нам из описаний работы протоколов Ethernet, используемых сетевыми картами локальных компьютерных сетей.

Активное программное обеспечение одного из узлов (вероятно компьютера) может захотеть попытаться принудительно выбрать один из узлов, пригодных в данной топологии шины, на роль вершины дерева (root node). В этом случае имеется возможность послать этому узлу пакет физического уровня (PHY) с соответствующим флажком и устанвить внутри этого узла параметр root delay в диапазоне от 83 до 167 микросекунд. Тогда после программной переинициализации шины шансы этого узла стать вершиной существенно увеличиваются.

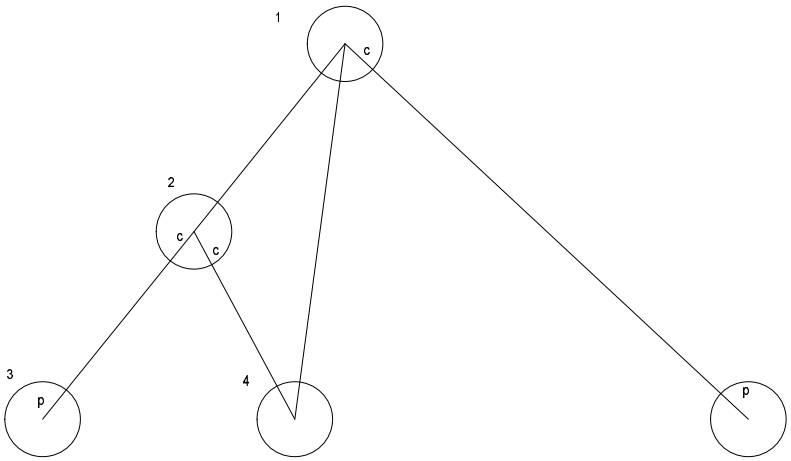
Если в процессе tree identification в течении интервала времени более 167 микросекунд узлы не могут пределить своих родителей – это означает что в топологии есть петли и в узлах, обнаруживших это условие обнаруживается флажок loop, тогда локальные приложения в этих узлах (будь те узлы видеокамерами или PC) могут сообщить пользователю о том, что шина не может быть сконфигурирована из за этих петель.

Рисунки иллюстрируют что при описаном способе определения топологии условие когда как минимум 2 узла не могут определить своих родителей возникают именно при наличии петель.





В случае если возникнет «кольцевая топология”, как на рисунке:



узлы 1 и 4 так и не смогут послать сигнал «parent notify” до истечения таймаута, в результате процесс tree identification так и не сможет завершиться, а за ним в свою очередь и sefl identification. В итоге шина работать не сможет!

Self identification

В результате процедуры Tree identification происходит локальное определение топологии шины, то есть устанавливаются связи соседних узлов и ля каждого узла задается направление к вершине дерева в сторону одного из соседних узлов.

В результате же следующей за ней следующей процедуры Self identification происходит: определение глобальной топологии (сохраняется и делается доступной для всех узлов карта текущей топологии шины), каждому узлу присваивается адрес называемый Physical ID, в процессе обмена соседними узлами информации о поддерживаемой ими максимальной скорости передачи создается “карта скоростей” соответствующая текущей топологии.

Адреа, присвоенные устройствам используются для идентификации устройств в процессе передачи данных через шину.

В силу того, что к шине могут быть подключены устройства, поддерживающие разные скорости передачи, обмен происходит на самой низкой скорости, определенной стандартом для FireWire устройств 100Mbit/sec.

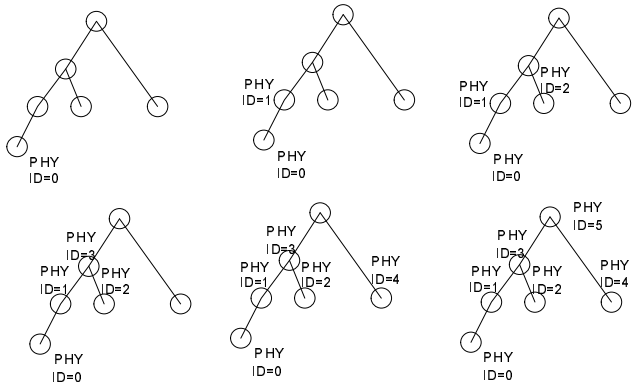
После завершения процедуры шина оказывается способной передавать данные между любыми двумя устройствами со максимальной скоростью определяемой самым медленным участком, соединяющим эти два устройства. В силу процедуры Tree initialisation для любых двух устройств на успешно сконфигурированной шине может существовать не более одного пути их соединяющего. Все устройства поддерживают одинаковую максимальную допустимую скорость для каждого из своих портов. Таким образом возможная скорость для любого соединения, проходящего через несколько устройств фактичекки определяется самым медленным из этих промежуточных устройств. Этот факт следует принимать во внимание для того, чтобы соединить имеющиеся устройства оптимальным способом. Понятно что может оказаться что при “неправильном” подключении некоторые соединения могут оказаться “недостаточно быстрыми” для того, чтобы уместить все логические каналы передачи. Если одним из устройств является компьютер, то пользователю могут быть доступны специальные программы визуализирующие “карту скоростей” шины. Такие программы могут быть полезны для того чтобы понять как же лучше соединить имеющиеся устройства.

Процесс назначения устройствам адресов происходит следующим образом. Узел, являющийся вершиной дерева (Root node), выбирает один из своих портов (имеющий самый маленький номер из всех подключенных портов) и при помощи арбитражного сигнала (TPA=”Z”, TPB=”0”). Дочерний узел, получив этот сигнал - делает то же самое, и в результате процесс достигает одного из листьев дерева, которому уже некому передавать сигнал. Этот последний узел присваивает себе значение адрес равное нулю и передает пакет данных, называемый Self ID packet. Этот пакет в числе прочего содержит адрес, присвоенный себе устройством и передается всем узлам шины. Благодаря этому все узлы определяют какой самый последний номер уже использован в качестве адреса (Physical ID) для одного из устройств.

Для продолжения процедуры, узел только что присвоивший себе номер, должен быть в дальнейшем исключен из списка опрашиваемых узлов (то есть узлов, которым родительский узел может направить сигнал TPA=”Z”, TPB=”0”, см. выше по тексту). Это достигается следующим способом: после завершения передачи Self ID packet узел присвоивший себе адрес передает своему родительскому узлу арбитражный сигнал Identification done (TPA=”1” TPB=”Z”), получив этот сигнал родительский узел знает что пославший его дочерний узел и вместе с ним все узлы, составляющие нижележащую ветвь дерева, уже присвоили себе адреса Physical ID. Ни на одном из следующих шагов процедуры

родительский узел уже не будет посылать запросов в эту ветвь дерева.

Далее процесс повторяется аналогичным образом. Родительский узел только что пронумерованного узла выбирает следующий подключенный порт, из которого еще не приходило сигнала Identification done и посылает в него соответствующий запрос. И лишь когда все порты исчерпаются присваивает адрес самому себе и сообщает на этот раз уже своему родитель что его ветвь полностью пронумерована. Таким образом в результате обхода дерева “с проваливанием до низа” каждому узлу (устройству) назначается свой адрес. Последним адрес получает корневой узел.



Каждый из узлов сразу же после посылки сигнала Identification done посылает своему родительскому узлу информацию о максимальной поддерживаемой им скорости и получает от него в ответ аналогичную информацию.

Впоследствии узлы которые будут выполнять роли менеджера шины (Bus manager) и менеджера изохронных ресурсов (Isochronous resource manager) на основании информации, собранной из пакетов переданных различными узлами в процессе процедуры Self identification сделают доступными для других узлов “карту скоростей” и схему топологии шины.

Self ID пакеты так же включают в себя информацию о портах устройства и имеют фиксированную длину (32 бита), поэтому, устройства, имеющие много портов могут передавать сразу несколько Self ID пакетов: устройства с числом портов от 4 до 11 передают 2 пактеа, а устройства с числом портов от 12 до 16 (максимально возможного) передают 3 пакета. При этом все последующие пакеты, имеют структуру, иную чем самый первый пакет, называемый (Self ID Packet Zero).

В MSB представлении (сначала тдут старшие биты, а потом младшие) 32-битный пакет Self ID Packet Zero имеет следующую структуру.

10 (2 bit)	Идентификатор пакета (PID) для Self ID пакетов всегда имеет такое значение
Physical ID (6 bit)	Адрес, которы присвоил себе данный узел (устройство)
0 (1 bit)	Для Self ID Packet Zero всегда 0 (а для следующих пакетов Self ID Packet всегда 1)

Link active (1 bit)	Флажок, устанавливаемый в зависимости от того, включены ли Link и Transaction уровни, данного устройства
Gap count (6 bit)	
PHY speed (2 bit)	00 100Mbit/sec; 01 100Mbit/sec and 200Mbit/sec; 10 100Mbit/sec, 200Mbit/sec, 400Mbit/sec; 11 extended speed, точную информацию можно считать из PHY регистров данного устройства.
PHY delay (2 bit)	
Contender (1 bit)	Если этот флаг установлен Link active, то данный узел подходит для выполнения роли менеджера шины или менеджера изохронных ресурсов
Power class (3 bit)	000 узел не использует питание из шины и не передает его в шину 001 узел имеет собственное питание и может отдавать в шину не менее 15 вт. 010 узел имеет собственное питание и может отдавать в шину не менее 30 вт. 011 узел имеет собственное питание и может отдавать в шину не менее 45 вт. 101 зарезервировано для использования в следующих реализациях 110 узел использует питание шины и для того чтобы его Link уровень мог работать узлу требуется не более 3 вт. 111 узел использует питание шины и для того чтобы его Link уровень мог работать узлу требуется не более 7 вт, при этом еще 3 вт. Могут потребляться при выключенном Link уровне.
Port numbers p0,p1,p2 (6 bit)	Для каждого из портов 2 бита определяют его состояние. 11 подключен к дочернему узлу 10 подключен к родительскому узлу 01 не активен (не подключен, запрещен и.т.п.) 00 порт отсутствует
Initiated reset (1 bit)	Установлен если данный узел инициировал текущую переинициализацию шины (Bus reset)
More packets (1 bit)	Установлен если за этим Self ID packet следуют еще другие.

Пакеты Self ID с номерами 1 и 2 имеют сходную структуру соответственно:

10	Physical ID (6 bit)	1	N (3 bit), номер пакета 1	r (2 bit)	p3...p10 (16 bit) состояния портов	I (1 bit)	M (1 bit)

10	Physical ID (6 bit)	12	N (3 bit), номер пакета	r (2 bit)	p11...p15 (16 bit)0состояния портов	Зарезервировано (8 bit)

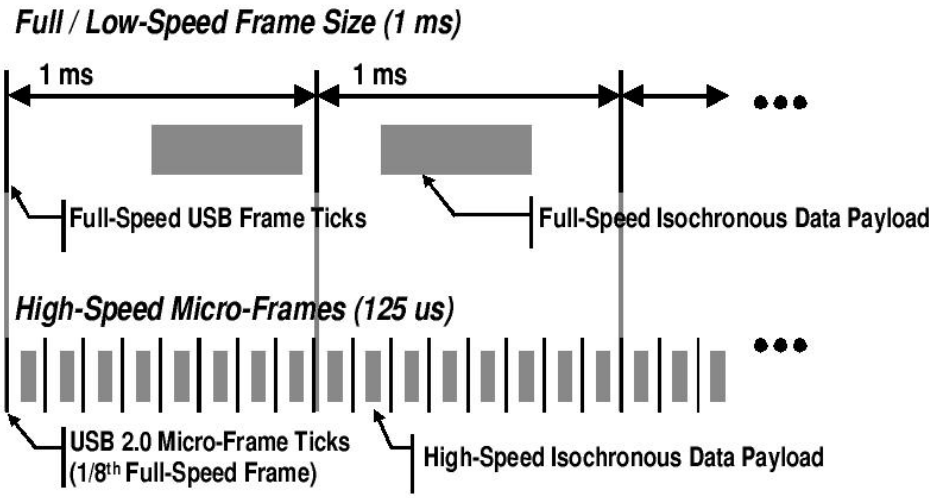
Передача пакетов

USB.

Таймслоты, совместимость стандартов, транзакции.

Передача всех пакетов по шине USB, как уже говорилось, происходит по команде хоста, начало каждого таймслота, называемого frame, маркируется посылкой хостом специального пакета SOF (statrt of frame). Подключенные к шине устройства могут использовать SOF для коррекции своих внутренних часов. Длительность таймслота (фрейма) USB1.0 составляет 1 миллисекунду, таймслоты USB2.0 принято называть микрофреймами (microframe), их длительность составляет 125 микросекунд. Для достижения совместимости обоих стандартов HUB-устройства USB2.0 и используют механизм конверсии транзакция при передаче данных между хостом и устройствами USB1.0 (то есть устройствами использующими скорости low spee и full speed).

Рисунок (USB farames и microframes)



Хост инициирует передачу данных устройствам (OUT) или от устройств (IN) при помощи специального пакета, называемого Token-пакет. Этот пакет содержит адрес устройства которое должно получить или передать данные и номер логического канала (endpoint-a) внутри этого устройства. Устройство может подтвердить готовность при помощи пакета ACK (acknowledge) или сообщить хосту том, что оно еще не готово для передачи или приема очередного пакета. В случае если требуется получить данные от устройства (IN-транзакция), хост может попытаться сделать это еще раз позднее. При передаче данных от хоста к устройству (OUT-транзакция) если по причине неготовности принять данные устройство вернет NACK, хост должен будет повторить псосылку пакета через некоторое время. В случае USB1.0 если устройство окажется неготово несколько раз подряд, каждый раз хост будет передавать последний непринятый этим устройством пакет, тем самым занимая время от таймслота, которое могло бы быть использовано для связи с другим устройствами! В USB2.0 эта проблема решается при помощи Ping-пакетов, позволяющих узнать готово ли устройство к приему данных для выбранного логического канала без передачи самих данных. Эта особенность относится в первую очередь к асинхронной передаче bulk, для изохронной передачи (isochron) в силу того, что нет подтверждения передачи (handshake) подобная ситуация возникнуть не может.

Низкоскоростные устройства (low speed 1.5Mbit/sec) могут использовать только control и interrupt транзакции, которые в целом ведут себя аналогично OUT и IN транзакциям типа bulk. При передаче пакетов для более быстрых устройств (full speed USB1.0 и high speed USB2.0) HUB-устройства не передают эти пакеты через порты, к которым подключены low speed устройства. При этом когда передаются данные для самих low speed устройств в составе пакетов используется еще специальный префикс, позволяющий HUB-ам определить что пакет следует ретранслировать на низкой скорости.

Подтверждение передачи происходит при помощи handshake пакетов: ACK, NACK, STALL. Хост никогда не может послать устройству NACK, вместо этого он просто не посылает никакого handshake пакета! Устройство может вернуть пакет STALL если произошла ошибка, требующая вмешательства хоста для переинициализации какого-либо ресурса в этом устройстве, связанного с логическим каналом в устройстве, для которого предназначался пакет.

Структура и типы используемых пакетов

Все пакеты начинаются с поля SYNC, которое служит для того, чтобы устройства могли синхронизировать свой внутренний счетчик битов с передаваемым пакетом. Для этого используется последовательность, содержащая максимальное количество переходов между уровнями логических нуля и единицы. Для используемой NRZI кодировки при передаче байтов (подробнее в главе про электрические характеристики) в качестве такой последовательности используется строка “KJKJKJKK”.

Кроме того каждый пакет в зависимости от типа (и соответственно назначения) может содержать следующую информацию: 4-х битовый идентификатор пакета (PID), семибитный адрес устройства, четырехбитовый номер логического канала (endpoint), данные содержимого пакета, контрольную сумму CRC, одиннадцатибитовый циклический номер фрейма (только для SOF пакетов).

Идентификатор пакета всегда следует за SYNC и является обязательным атрибутом пакета. Вслед за PID предается его собственная (независимая от CRC) четырехбайтовая контрольная сумма. В зависимости от типа пакета и версии реализации USB, возможные значения PID перечислены в таблице.

Тип пакета	***Название PID***	***Значение PID***	***Описание***
Token	OUT	0001B	Адрес + номер endpoint-а при передаче в устройство
	_	_	Адрес + номер endpoint-а при передаче из устройства
	IN	1001B	
	_	_	Маркер начала фрейма + номер фрейма
	SOF	0101B	
	_	_	Адрес + номер endpoint-а для транзакции типа control
	SETUP	1101B	

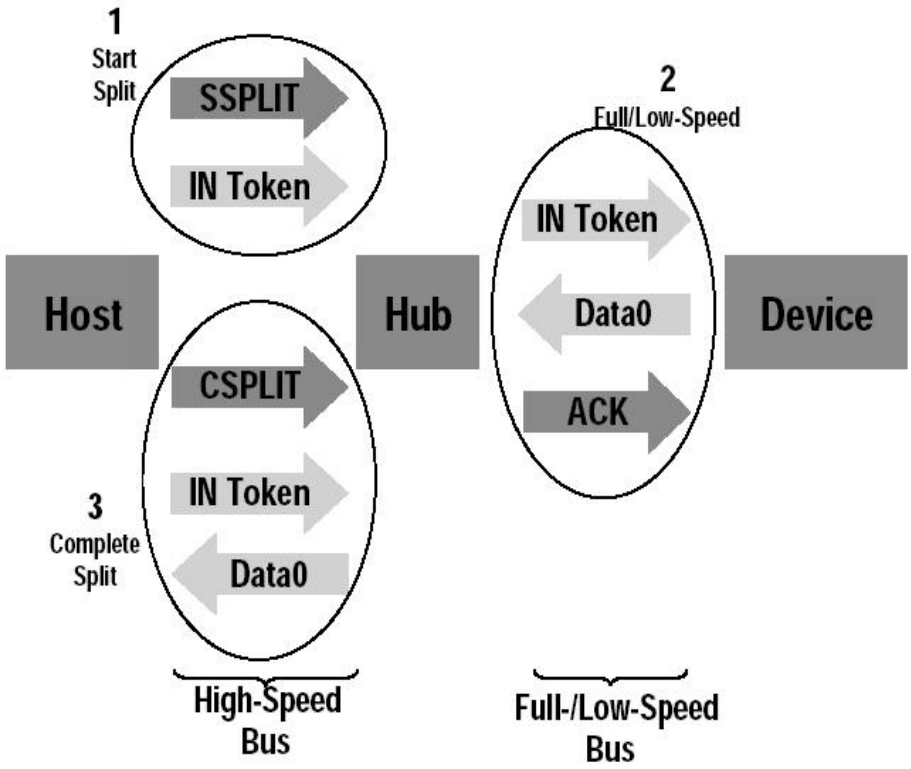
Тип пакета	*Название PID*	*Значение PID*	*Описание*
Data	DATA0	001B	Пакет данных счетным номером.
	_	_	
	DATA1	1011B	Пакет данных с нечетным номером.
	_	_	
	DATA2	0111B	Данные high speed транзакции типа bulk или isochron в микрофрейме.
	_	_	
	MDATA	1111B	То же, включая split-транзакции.
Handshake	ACK	0010B	Данные получены без ошибок
	_	_	
	NACK	1010B	Устройство не готово принять или послать данные
	_	_	
	STALL	1110B	Устройство остановило передачу т.к. Требуется вмешательство для устранения ошибки.
	_	_	
	NYET	0110B	Устройство еще не готово передать данные (при работе с ping пакетами USB2.0)
Special	PRE	1100B	Префикс передачи для low speed устройств
	ERR	1100B	(значение совпадает с ERR)
	SPLIT	1000B	Handshake ошибка для split тарнзакций.
	PING	0100B	USB2.0 handshake пакет для проверки готовности устройств передать данный типа bulk/control
	Зарезервировано	0000B	

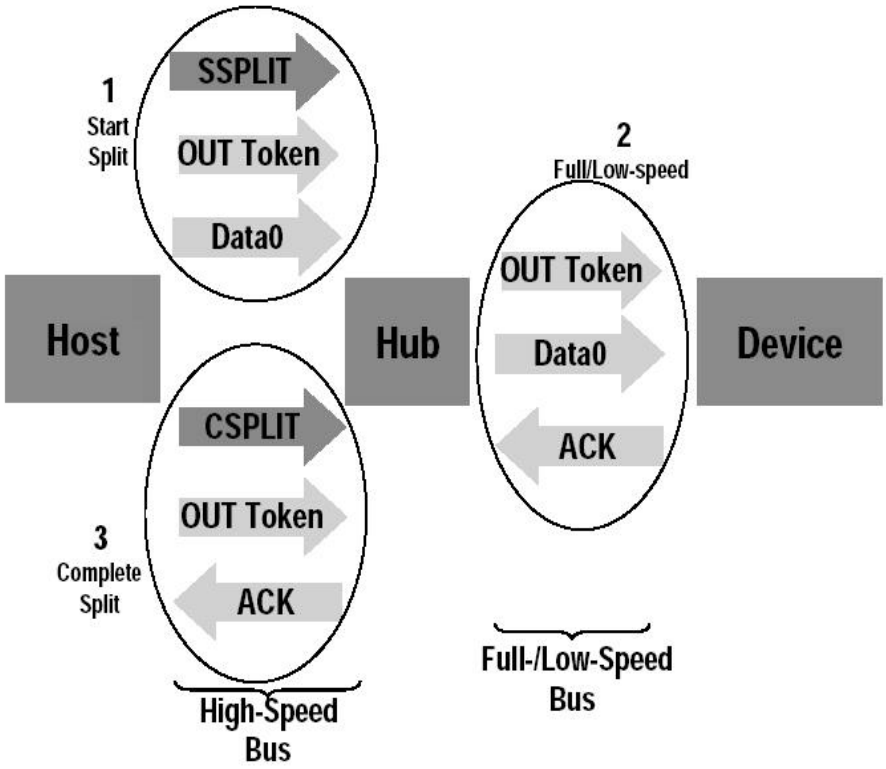
Split-транзакции

Split-транзакции в USB предназначены для использования между хостом и HUB устройствами. Транзакции этого типа используются в качестве вспомогательного механизма, когда устройства full-speed или low-speed подключены к HUB, работающему в режиме high-speed. Суть split-транзакции состоит в том, что запрос и ответ разделены во времени, тем самым оставляя возможность шине использовать свою пропускную способность между запросом и ответом для обслуживания других устройств (механизм split-транзакций существует так же и для FireWire).

При работе с медленным (low-speed или high-speed) устройством хост может начать любую передачу для этого устройства при помощи split-start команды для HUB, к которому подключено данное устройство. При этом поскольку ответ от устройства может поступить существенно позже (по меркам скоростей передачи USB2) хост в последствии может получить ответ от устройства при помощи split-complete команды для HUB. Существование split-транзакций таким образом незаметно для всех устройств кроме HUB-ов, поддерживающих high-speed передачу.

Рисунки (split-транзакции для IN и OUT направлений)





Описание транзакций для разных типов передачи данных USB (bulk, isochron, control, interrupt)

Для каждого типа передачи выделяют два типа транзакций IN (передача пакетов от устройства к хосту) и OUT (передача данных от хоста к устройству). Однако в зависимости от типа (bulk, isochron, control, interrupt) протокол обмена пакетами данных может существенно отличаться. Рассмотрим подробнее каждый из типов.

Bulk транзакции

Bulk обеспечивает передачу данных с подтверждением и повторной передачей в случае возникновения ошибок (гарантированная доставка). При этом данные передаются потоком один пакет вслед за другим. Каждый bulk endpoint в устройстве имеет тип IN либо OUT и соответственно предназначен для передачи данных только в одном направлении. Ответ NACK от устройства на token пакет означает что пакет получен без ошибок оно еще не готово к приему или передаче данных и поэтому хосту следует повторить попытку через некоторе время. Соответственно ответ NACK на пакет с данными от хоста (OUT) означает

что данные хост должен послать данные еще раз несколько позже. Если устройство довольно медленное, то последнее может приводить к неоправданному расходу пропускной способности шины! Поэтому в USB2.0 предусмотрен специальный тип пакетов ping, предназначенных для проверки состояния готовности устройства. Благодаря этому механизму, хост может не производить повторную посылку пакета данных пока не получит информацию о готовности устройства.

В случае если какой-либо пакет требует повторной передачи (ответный handshake пакет не получен) хост может попытаться произвести повторную посылку или прием пакета. Ответ handshake STALL означает что произошла ошибка, требующая вмешательства ПО для восстановления состояния устройства. При успешном получении пакета данных хост так же отвечает устройству ACK.

Процедура передачи пакета (как мы видели) делится на три стадии: token, data и handshake. Весь этот механизм обеспечивает гарантированную доставку пакетов в течении не гарантированного времени. На практике это означает что ошибки передачи выливаются в увеличенное время, поэтому устройства кторым необходим непрерывный поток данных не содержащих ошибки (например такие как устройства зсписи CDROM) должны иметь достаточно большой внутренний буфер для компенсации временных задержек, кроме того они должны стараться держать этот буфер заполненным (в частности начинать процесс записи только после того, как получат от хоста достаточный объем данных). Кроме того, на примере CDR/W, скорость записи на диск должна быть не слишком высокой чтобы подкачка оставшихся данных в буфер могла быть произведена за время записи данных, уже находящихся в буфере. Понятно что выполнение этих условий сильно зависит от скорости передачи, общего количества передаваемых данных и объема буфера и в принципе поддается оценке в каждом конкретном случае. Поэтому если мы хотим избежать “игры в Русскую рулетку” нам следует выбрать скорость записи ниже средней скорости передачи bulk через шину USB. Последнее, в силу того что каналы bulk не имеют выделенной специально под каждый канал полосы пропускания на шине а конкурируют за нее в процессе работы, вообще говоря не избавляет нас от возможных проблем, связанных с наличием других устройств (которые могут в самый неподходящий момент захотеть передать большой объем информации). Тут невольно хочется вспомнить времена старого доброго однозадачного DOS.

Следующий рисунок иллюстрирует транзакции типа bulk.

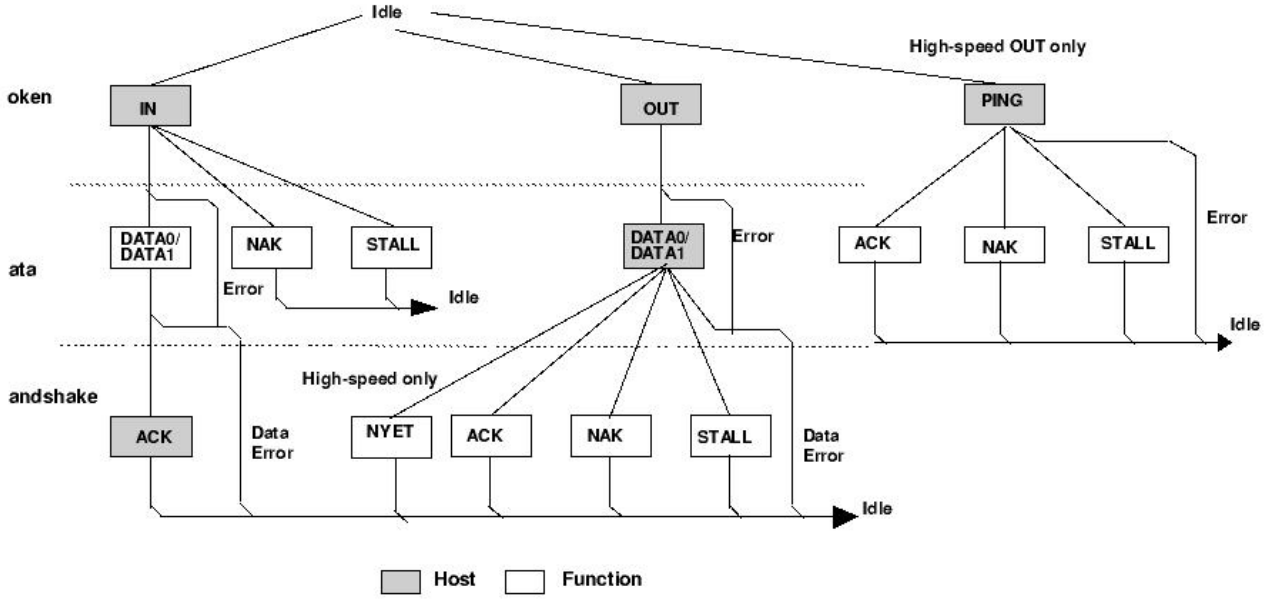


Рисунок (bulk транзакции)

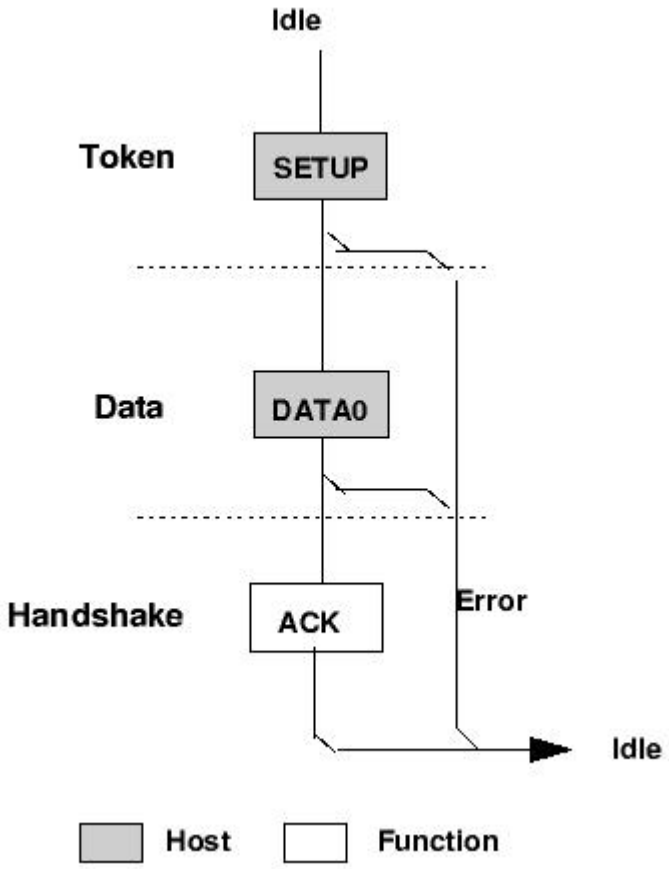
Control транзакции .

Транзакция control состоит из трех этапов: setup, data и status. Хост начинает транзакцию посылкой пакета SETUP, при этом передается информация о направлении передачи (IN или OUT) и предполагаемое количество байт данных. На этапе data (таковой может отсутствовать если нет данных для передачи) передаются данные в выбранном направлении, возможно количество данных будет больше заявленного и тогда этот этап будет содержать один или несколько дополнительных пакетов данных. Третий этап staus – передача пакета, содержащего ответ, в обратном направлении. Завершает весь процесс обычная handshake процедура (NACK означает что устройство не готово, а STALL означает ошибку).

Устройство, получившее token пакет SETUP в котором указан номер default endpoint (или номер любого другого control endpoint-a) в качестве номера endpoint-a обязано принять этот пакет (handshake ответы от устройства NACK и STALL не предполагаются, в случае ошибки устройство не возвращает никакого handshake пакета), если в качестве номера указан номер endpoint-a неподходящего типа, устройство должно проигнорировать пакет. Передача может быть разбита на несколько пакетов, но при этом IN и OUT пакеты никогда не смешиваются пока не закончится транзакция (за исключением пакета, передаваемого а этапе status). Фактически это означает что хост не потребует от устройства передачи данных (IN) через control endpoint до тех пор пока сам не завершит передачу (OUT).

Тип передачи control предназначается не для потоковой, а для пакетной передачи, поэтому пакеты control имеют определенную стандартом структуру, включающую информацию о направлении передачи, назначении пакета, длине вложенных данных и.т.п. Существует целый ряд служебных команд, которые могут быть переданы через default endpoint.

Рисунок (control транзакции)



В таблице представлена структура заголовка пакета control. Битовые структуры представлены в виде D7,D6...D0, где D7 старший бит.

Смещение в байтах	***Длина поля***	***Тип***	***Название поля***	***Структура***
0	1	Битовая маска	bmRequestType	D7:направлениепередачи 1 OUT 0 IN D6D5: тип 00 Standard 01 Class 10 Vendor 11 Reserved D4D0:получатель
1	1	значение	bRequest	Специфический request, подробнее в следующих таблицах.

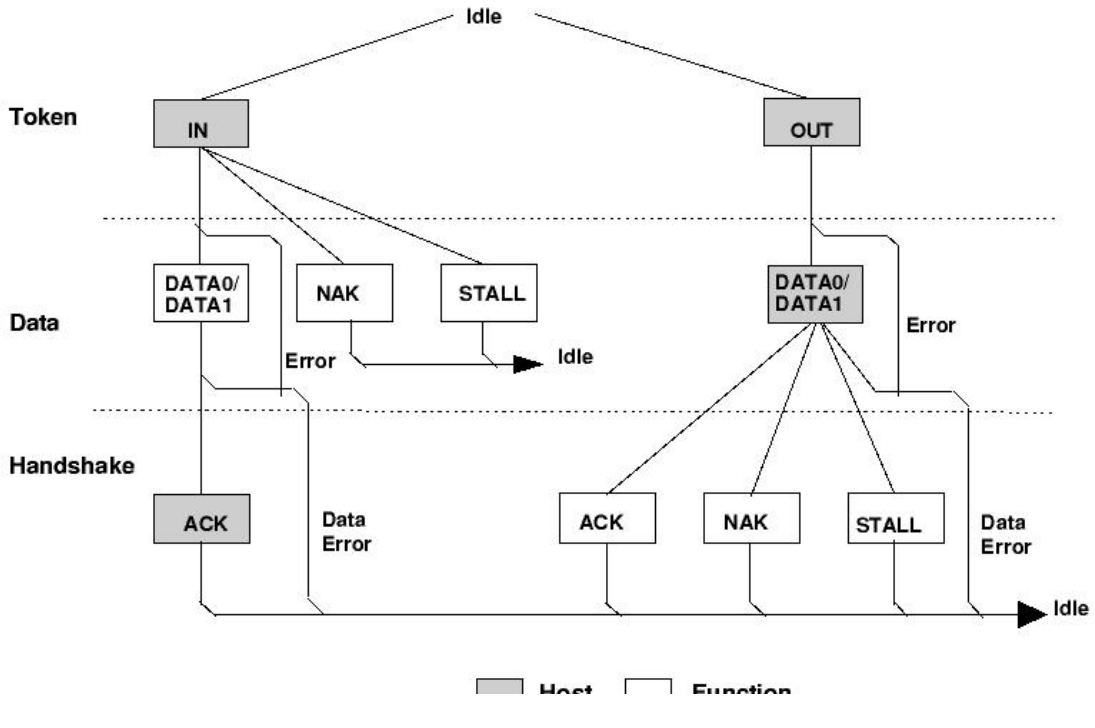
Смещение в байтах	*Длина поля*	*Тип*	*Название поля*	*Структура*
2	2	значение	wValue	Зависит от bRequest
4	2	индекс	wIndex	Зависит от bRequest (обычно некоторое смещение)
6	2	Количество байтов	wLength	Количество передаваемых байт данных

Полностью структура control приведена раньше. Именно названия полей из этой таблицы и фигурируют в качестве названий же аргументов функций программного API для работы с шиной USB. Важно уметь заполнять их при работе с устройством согласованным образом, поскольку работа программы (или драйвера) с устройством как правило начинается с посылки пакета в default endpoint устройства. При обнаружении шиной устройства, драйвера нижнего уровня поддержки USB операционной системы получают используют default endpoint для первоначального конфигурирования устройства и получения различных сведений о нем, которые после могут быть предоставлены приклодной программе или драйверу.

Interrupt транзакции.

Хост посылает IN-token и устройство в ответ возвращает либо пакет данных, либо NACK (если данные не готовы), либо STALL (если данные не готовы). Получив пакет хост так же отвечает устройству ACK.

Процедура аналогична bulk транзакции типа IN, включая гарантированную доставку данных от устройства к хосту. Таким образом interrupt транзакция предназначена для передачи небольшого количества данных, например для использования в качестве канала



обратной связи например при передаче данных по изохронному каналу. При этом поскольку хост опрашивает устройство с частотой, которое указало само устройство, interrupt передача может оказаться более оперативной чем bulk.

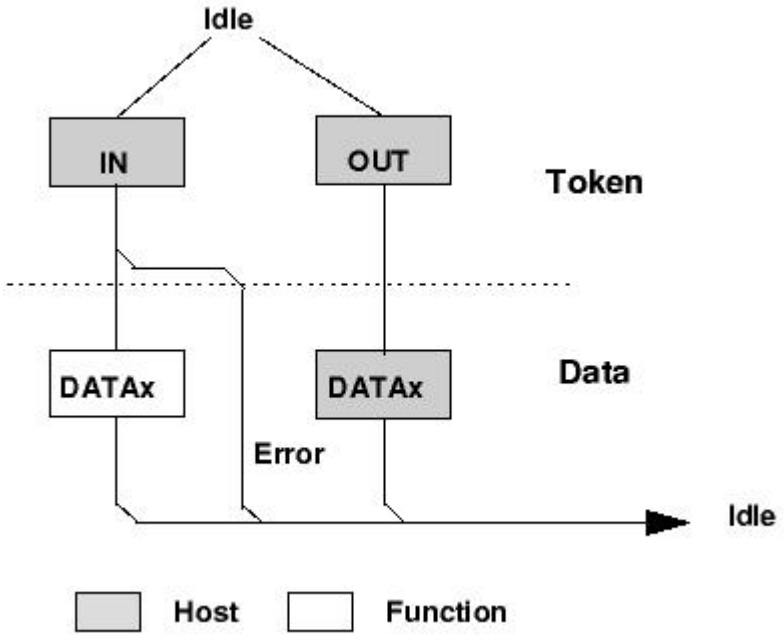
Для lowspeed устройств USB1.0 разрешенными типами передачи являются control и interrupt и поэтому производители устройств нередко используют interrupt для асинхронной передачи данных от устройства к хосту. Примером такого устройства может служить ридер чиповых карт ACE30U (производства Advanced Card Systems), для передачи данных от хоста к устройству в данном случае используется control передача.

В более поздних версиях спецификации (USB1.1) добавлено понятие interrupt передачи типа OUT. Повидимому это было сделано для того, чтобы обеспечить более удобный способ работы для медленных устройств с тем, чтобы вместо единственного канала для передачи от хоста к устройству типа control можно было использовать и другие каналы. В частности такой способ может использоваться устройствами, реализующими поверх USB прикладной протокол HID. HID – переводится как Human Interface Dvice и формализует некоторые общие свойства медленных устройств типа мышей и джойстиков.

Isochron транзакции.

Хост передает устройству IN/OUT token пакет и вслед за ним принимает пакет данных isochron. Никакого handshake, никакой повторной передачи. Доставка данных не гарантирована, однако хост инициирует передачу пакетов с достаточной частотой и регулярностью, что каждый изохронный канал получает определенную (заявленную устройством) полосу пропускания для передачи. При этом максимальная латентность при передаче определяется только длительностью таймслота, которая составляет 1 миллисекунда для USB1.0 и 125 микросекунд для USB2.0.

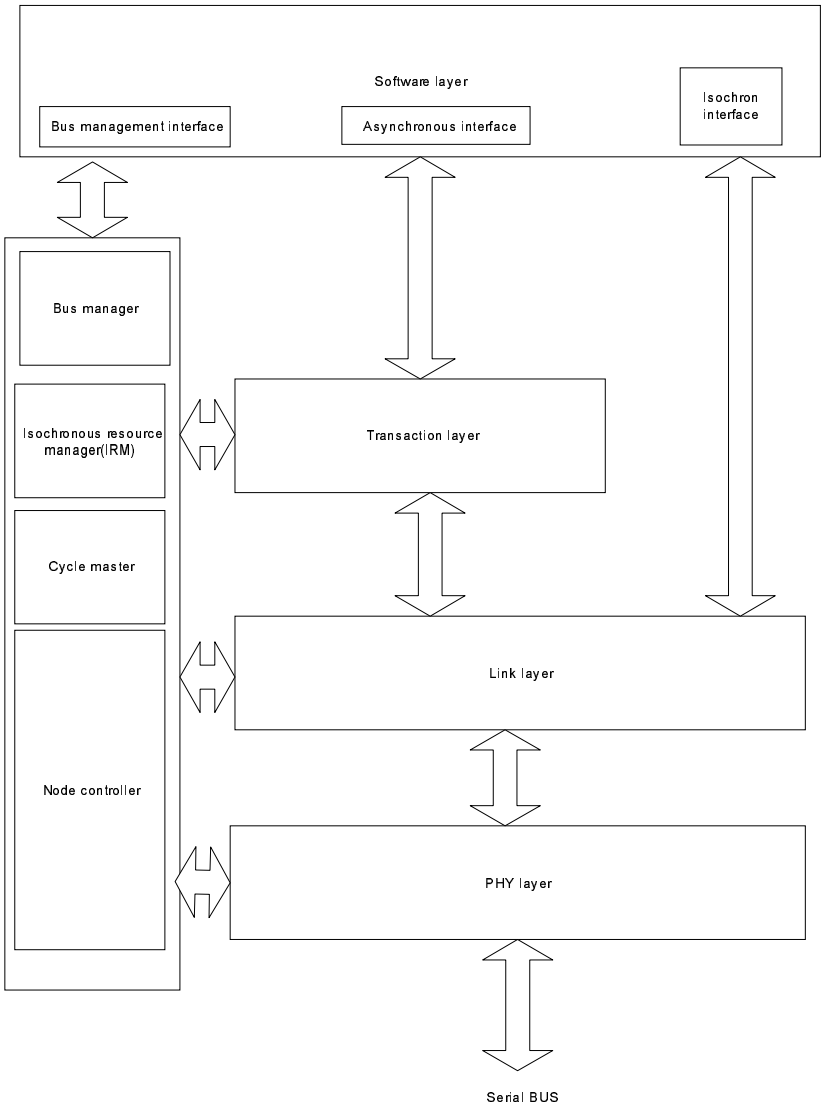
Рисунок (isochron транзакции)



FireWire.

Уровни протокола FireWire

Спецификация FireWrare определяет для протокола обмена данными следующие уровни:
1.Программный (Software diver)
2.Уровень транзакций (Transaction layer)
3.Уровнь звена связи (Link layer)
4.Уровень физического канала связи (Physical layer)



Уровень транзакций (Transaction layer) в случае асинхронной передачи данных обеспечивает гарантированную доставку пакетов за счет получения передающим

устройством подтверждения от принимающего, а так же за счет повторной передачи в случае возникновения ошибок.

При изохронной передаче гарантированной доставки не требуется и поэтому изохронные каналы используют для связи Link layer напрямую. Для Link layer и Physical layer существуют чипов, реализующие каждый из этих уровней. Электрический интерфейс между этими чипами так же определяется стандартом и носит название OHCI (Open Host Controller Interface) IEEE1394.

Рисунок (Логическая организация OHCI-1394)

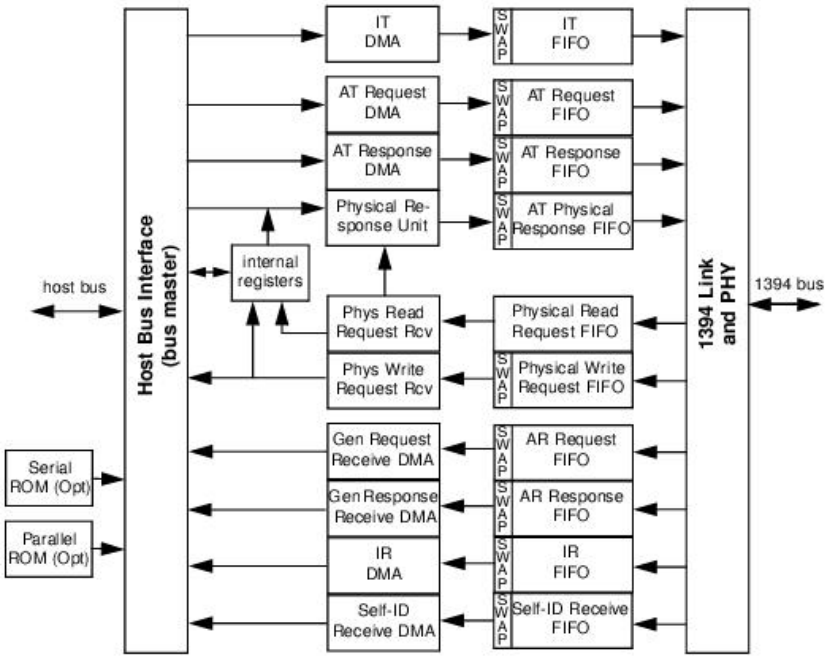


Figure 1-1 — 1394 Open HCI conceptual block diagram

Как уже говорилось во введении совокупность для FireWire в тличие от USB на логическом уровне используется не парадигма каналов связи а парадигма области памяти произвольного доступа, представляющей совокупность всех устройств. Поскольку так же в отличие от USB не существует выделенного узла (хоста), который бы отвечал за распределение всех ресурсов шины, возникает необходимость синхронизации при попытке одновременного доступа несколькими устройствами к одному и тому же ресурсу (например попытке одновременно аллокировать канал и часть полосы пропускания для изохронной передачи). Подобная ситуация требует от устройств неукоснительного соблюдения процедур, определяемых IEEE1394 и в значительной мере делает реализацию FireWire более сложной чем реализация USB. Собственно говоря не удивительно что производители “железа” FireWire часто указывают в составе конфигурации с какими другими устройствами тестировалось предлагаемое ими оборудование. Так же иногда в Интернет мы можем прочитать сообщения о несовместимости некоторых устройств.

Асинхронные транзакции FireWire бывают трех типов:

1.read – чтение по указанному адресу виртуального адресного пространства

2.write – запись по указанному адресу

3.lock – то же запись но с проверкой соответствия предыдущего состояния записываемой области некоторой указанной. (на самом деле существует несколько вариантов транзакций типа lock, включающих различные режимы сравнения и хаписи данных, например с использованием битовых масок)

Первые два способа используются для обычной передачи данных между

устройствами, в то время как третий предназначен для использования в тех случаях когда необходима синхронизация при одновременном доступе к разделяемому ресурсу такому, как например как полоса для изохронной передачи.

Транзакция типа lock завершается успешно если в момент перед моментом записи текущее состояние записываемой области совпадает с некоторым указанным в той же транзакции состоянием, в противном случае запись не происходит. Такой способ как раз используется устройствами, когда они хотят изменить состояние таблицы содержащей зарезервированную полосу пропускания и номера аллокированных каналов, находящуюся в устройстве, выполняющем роль менеджера изохронных ресуррсов.

Полоса пропускания выделяется в терминах составляющих долей временного таймслота, длительность которого составляет 125 микросекунд. Как и для шины USB только примерно 80% полосы может быть выделено под использование изохронными каналами, весть таймслота разбивается на 6144 равных интервалов времени из которых 4915, составляющих вместе 100 микросекунд и могут быть задействованы под изохронные каналы. Общее число одновременно аллокированных каналов может достигать 64. При этом несколько устройств могут одновременно принимать данные, передаваемые по одному каналу в режиме broadcast, что может быть чрезвычайно полезно если, например, при передаче видеоизображения из некоторого источника на шине требуется одновременно производить его запись одним устройством и воспроизведение другим. Для сравнения в силу своей “централизованности” USB вообще не обладает такой возможностью, как и вообще возможностью обмена данными между устройствами напрямую (минуя хост).

Соблюдение правильности распределения времени во время одного таймслота между различными устройствами достигается за счет сочетания использования арбитража шины с использованием более коротких таймаутов перед попыткой захвата шины для передачи более приорететных пакетов чем таймаутов перед попыткой захвата шины для передачи более низкоприорететных пакетов.

Изохронные пакеты передаются с наибольшим приорететом, а для асинхронных действует “принцип справедливости”. Это означает что на протяжении 125 микросекундного таймслота все изохронные пакеты обязательно должны быть переданы, а оставшееся время может быть распределено между асинхронными пакетами таким образом что, если после того как каждое устройство, желавшее предать асинхронный пакет это сделает, останется дополнительное время, то все устройства смогут еще раз начать асинхронную передачу и так до конца таймслота.

Сценарий при помощи которого шина реализует эти правила – следующий:
Определяющцю роль в установлении порядка передачи данных различных типов (асинхронных и изохронных) устройствами играют велиичины временных задержек (таймаутов) которые начинают отсчитываться каждым из устройств в отсутствии передачи данных по шине, перечисленные в следующей таблице.

Acknowledge Gap	Время между посылкой асинхронного пакета и получением подтверждения ACK (Acknowledge), предполагается что веь процесс происходит атомарно. (0.04-00.5 микросекунд)
Isochronous Gap	Время до начала возможной попытки “захвата шины” устройством при помощи механизмов арбитража для посылки изохронного пакета, относящегося к изохронному каналу передачи. Это время короче чем Subaction Gap и чем обеспечивается более высокий приоритет передачи изохронных пакетов. (0.04-00.5 микросекунд)
Subaction Gap	Время до начала возможной попытки “захвата шины” устройством при помощи механизмов арбитража для посылки асинхронного пакета. (около 10 микросекунд в зависимости от сложности топологии шины)

Acknowledge Gap	Время между посылкой асинхронного пакета и получением подтверждения ACK (Acknowledge), предполагается что веь процесс происходит атомарно. (0.04-00.5 микросекунд)
Arbitration Reset Gap	Промежуток времени до наступления очередного промежутка fairness interval, в течении которого шина гарантирует что каждое устройство получит “справедливый” доступ к каналу для передачи асинхронного пакета. (20 микросекунд)

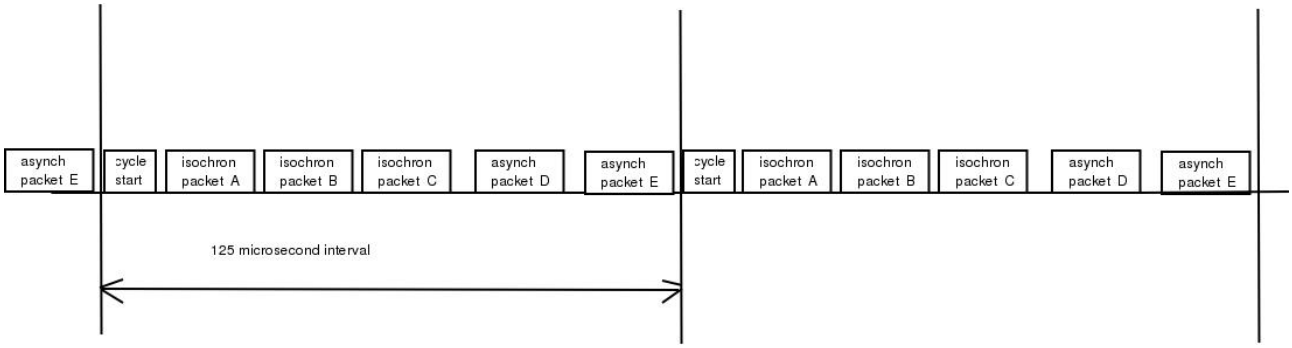
На протяжении одного таймслота (125 микросекунд) каждое устройство, желающее передать изохронный пакет имеет возможность это сделать. Для этого такое устройство, обнаружив отсутствие активности на шине (idle state) в течении интервала Isochronous Gap может произвести попытку “захвата шины” чтобы произвести собственную передачу. При этом поскольку устройства которые хотят произвести асинхронную передачу ожидают состояния шины idle state в течении более длительного интервала времени Subaction Gap, ни один асинхронный пакет не сможет быть переданным в течении текущего таймслота пока устройства желающие это сделать не передадут изохронные пакеты!

Важно то, что время отведенное под изохронные пакеты строго рассчитано таким образом что как минимум 25 микросекунд должно остаться для передачи асинхронных пакетов.

Для асинхронных пакетов fairness interval обеспечивает так называемую “ротационную схему приоритетов”, которая очевидно известна читателям знакомым с моделями приоритетов для нитей (threads) которая описывается в литературе по многопотоковому программированию (в частности направляем читателя к следующей литературе и интернет ресурсам [6], http://www.mikecramer.com/Qnx/momentics_nc_docs/neutrino/bookset.html, , http://www.peterindia.net/MultithreadProgramming.html, http://www.scis.nova.edu/~linje/cs770/mtweb.html, http://java.sun.com/docs/books/tutorial/essential/threads/index.html). В течении fairness interval каждое устройство пожелавшее передать асинхронный пакет может сделать это один лишь раз. Передав пакет, устанавливает внутри себя специальный флажок (Arbitration enable bit), запрещающий асинхронную передачу. Этот флажок сбрасывается когда по истечении Arbitration Reset Gap неактивности шины (idle state) устройства обнаруживает окончание fairness interval.

Иногда за счет того что асинхронный пакетб передача которого началась перед самым завершением 125 микросекундного интервала, может затянуться весть цикл передачи (это явление имеет название Cycle start skew). В этом случае в течении 1-2 последующих таймслотов задержка будет наверстана за счет уменьшения объема передаваемых асинхронных данных.

Рисунок (Иллюстрация порядка передачи пакетов по шине, как следствие арбитража)



Для “захвата шины” устройство посылает своему родительскому узлу сигнал TX_REQUEST этот сигнал распространяется по шине наверх в сторону вершины дерева и в конце концов устройство, выигравшее арбитраж, получает сверху ответ TX_GRANT. Во все остальные ветви дерева корневой узел передает сигнал DATA_PREFIX с тем, чтобы устройства в этих ветвях прекратили попытки “захватить” шину. По окончании этой процедуры выигравшее устройство может передать свой пакет. В таблице ниже указаны значания для используемых арбитражных сигналов.

Тип сигнала	***TPA***	***TPB***
TX_REQUEST	“Z”	“0”
TX_GRANT	“Z”	“0”
TX_DATA_PREFIX	“0”	“1”

Версия стандарта IEEE1394a определяет по сравнению с более ранней IEEE1394-1995 несколько ускоренных схем арбитража, которые используются для некоторых частных случаев передачи пакетов и позволяют увеличить эффективную пропускную способность шины:Acknowlwdge accelerated arbitration, Fly-by arbitration. За дополнительными подробностями отсылаем к текстам соответствующего стандарта а так же другой литературе [1].

Split-транзакции.

Так же как и USB, шина FireWire для работы с медленными устройствами предусматривает Split-транзакции. Суть этих транзакций в том, что медленное устройство, получив пакет read/write/lock вместо того, чтобы немедленно ответить на него соответствующим образом, может лишь подтвердить получение при помощи ACK. После завершения операции это медленное устройство может отдельно направить ответ устройству, обратившемуся к нему и получить от в свою очередь него обратно ACK на свой ответ. Посылка ответа происходит при этом на общих основаниях с использованием арбитража, поэтому между запросом и ответом шина может свободно использоваться другими устройствами. По сравнению с таковыми для USB транзакции типа Split выглядят даже несколько проще потому, что медленное устройство, которое должно прислать ответ само об этом “помнит” (в случае же USB, хосту нужно не забыть обратиться к устройству за ответом несколько позднее). Кроме того, устройство может начать пытаться передавать ответ сразу же как только он будет готов, естественно запросившему устройству (хосту в

случае USB этот момент точно не известен).

Рисунок (Иллюстрация механизма Split-транзакций)

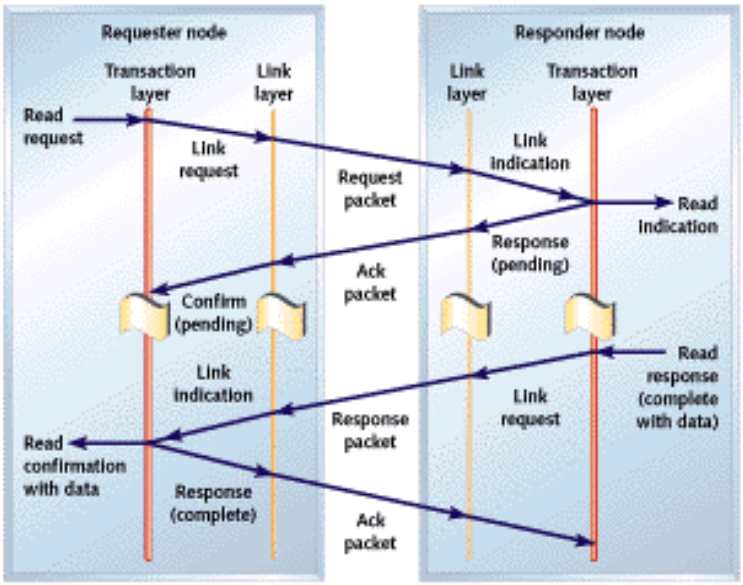
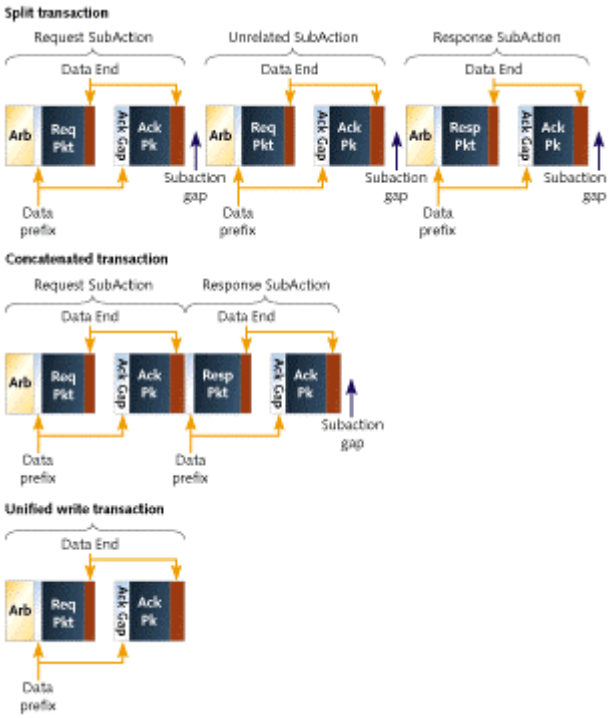


Рисунок (Типы транзакций FireWire)



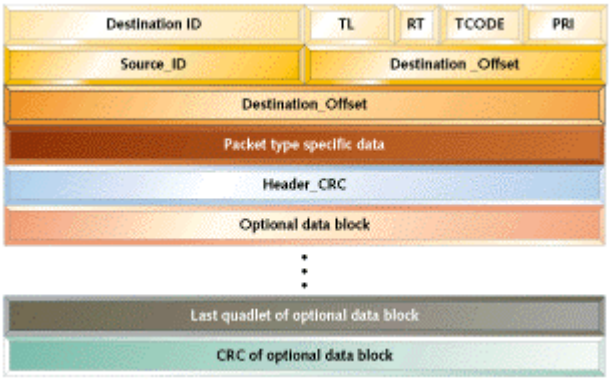


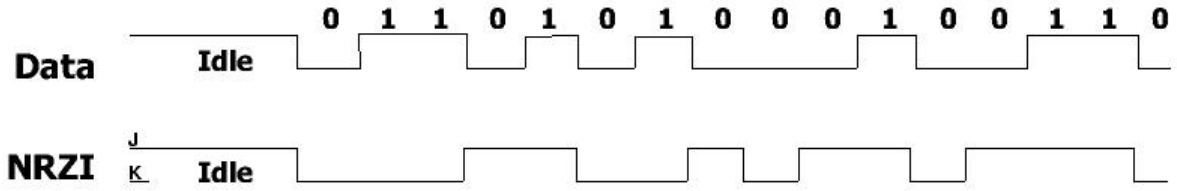
Рисунок (Пакеты транзакций FireWire)

Электрические характеристики, передача сигналов

USB.

Для передачи битов данных USB использует NRZI кодирование (для сравнения FireWire использует NRZ), смысл использования таких кодировок состоит в упрощении синхронизации передающих и приемных цепей устройств друг с другом. Подобные кодировки обеспечивают повышенное количество переходов между уровнями логического нуля и единицы по сравнению с непосредственной передачей битовой последовательности.

Рисунок (Кодирование NRZI)



При кодировании NRZ единица кодируется как переход между между уровнями 0 и 1, а ноль как отсутсвие перехода. Это похоже на передачу вместо сигнала его производной. По сравнению с этим, кодирование NRZI добавляет ноль после каждых встретившихся последовательно единиц. Для USB это особенно важно так как сигналы предаются по одному двухпроводному каналу, а канал для передачи строба отсутствует!

Для генерации контрольной суммы для данных и token используются 2 различных полинома, с использованием полиномов для кодирования можно ознакомиться в спциальной литературе: [4], [5].

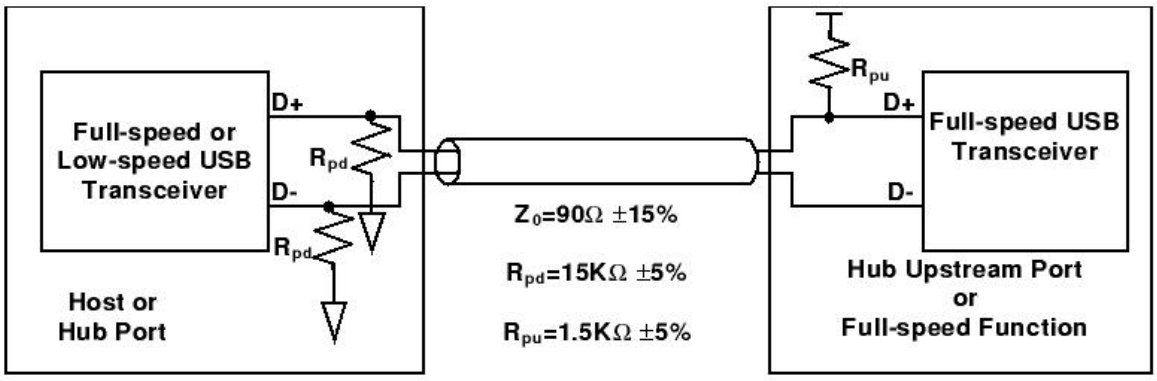
Мы не станем описывать детали реализации электрического протокола передачи USB, а лишь остановимся на некоторых основных особенностях. За более подробной информацией следует обращаться к стандарту USB, а так же интернет ресурсу www.opencores.org, где имеются исходные тексты реализации протокола передачи сигналов для устройств программируемой логики (ПЛИС) на языке VHDL. Имеются так же специализированные микросхемы, реализующие протоколы USB хоста и устройства, которые скрывают детали реализации внутри кристалла.

Передача данных происходит в полудуплексном режиме с использованием трех состояний, как правило ТТЛ логики, Приемные цепи измеряют дифференциальное напряжение на сигнальной паре проводов относительно локальной земли. Максимальная допустимая емкость линий относительно земли составляет порядка 10 пикофарад, время нарастания фронта импульса около 5 наносекунд. Напряжения сигнала достигают величин порядка 3.8 вольт. Передающие цепи не должны выходить из строя в случае если другая передающая цепь так же приложит такое напряжение к сигнальной паре проводов.

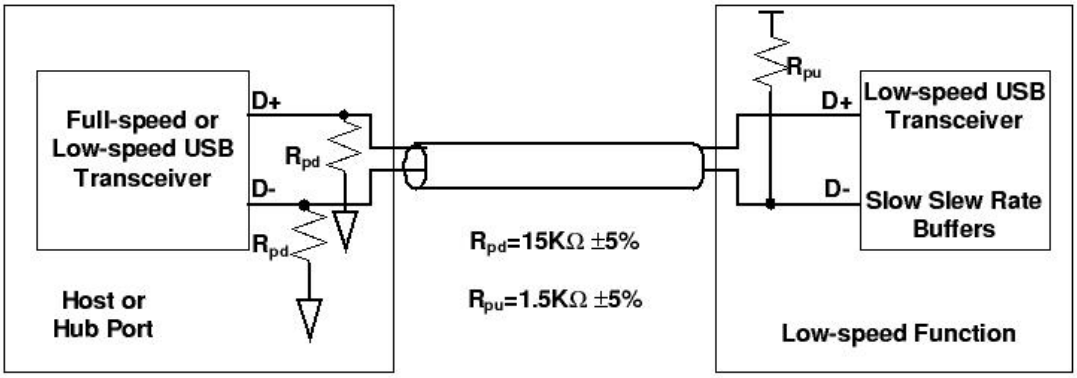
Максимальная длина соединительных проводов составляет около 5 метров

Рисунок (High speed и Low speed электричексие соединения)

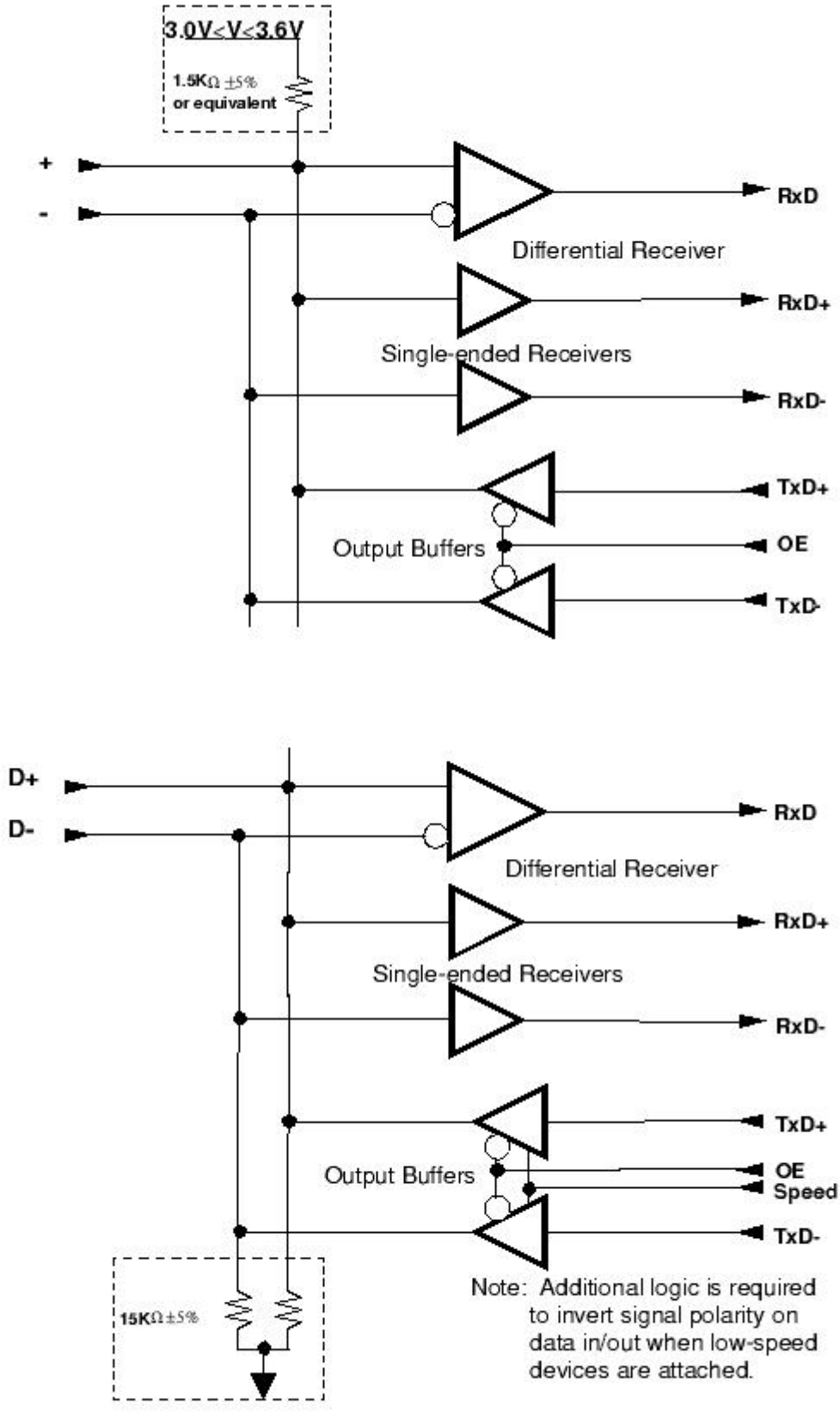
Рисунок (Пример схемотехники для USB шины)



Full-speed Device Cable and Resistor Connections



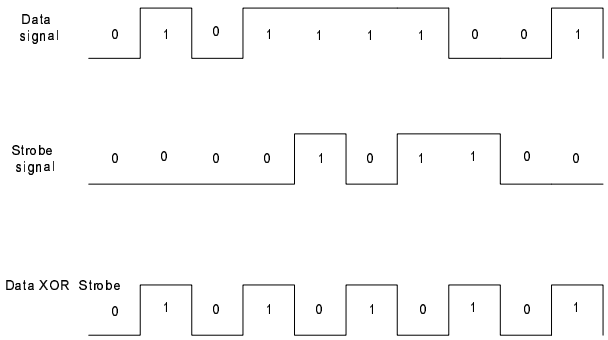
Low-speed Device Cable and Resistor Connections



Downstream Facing Low-/full-speed Port Transceiver

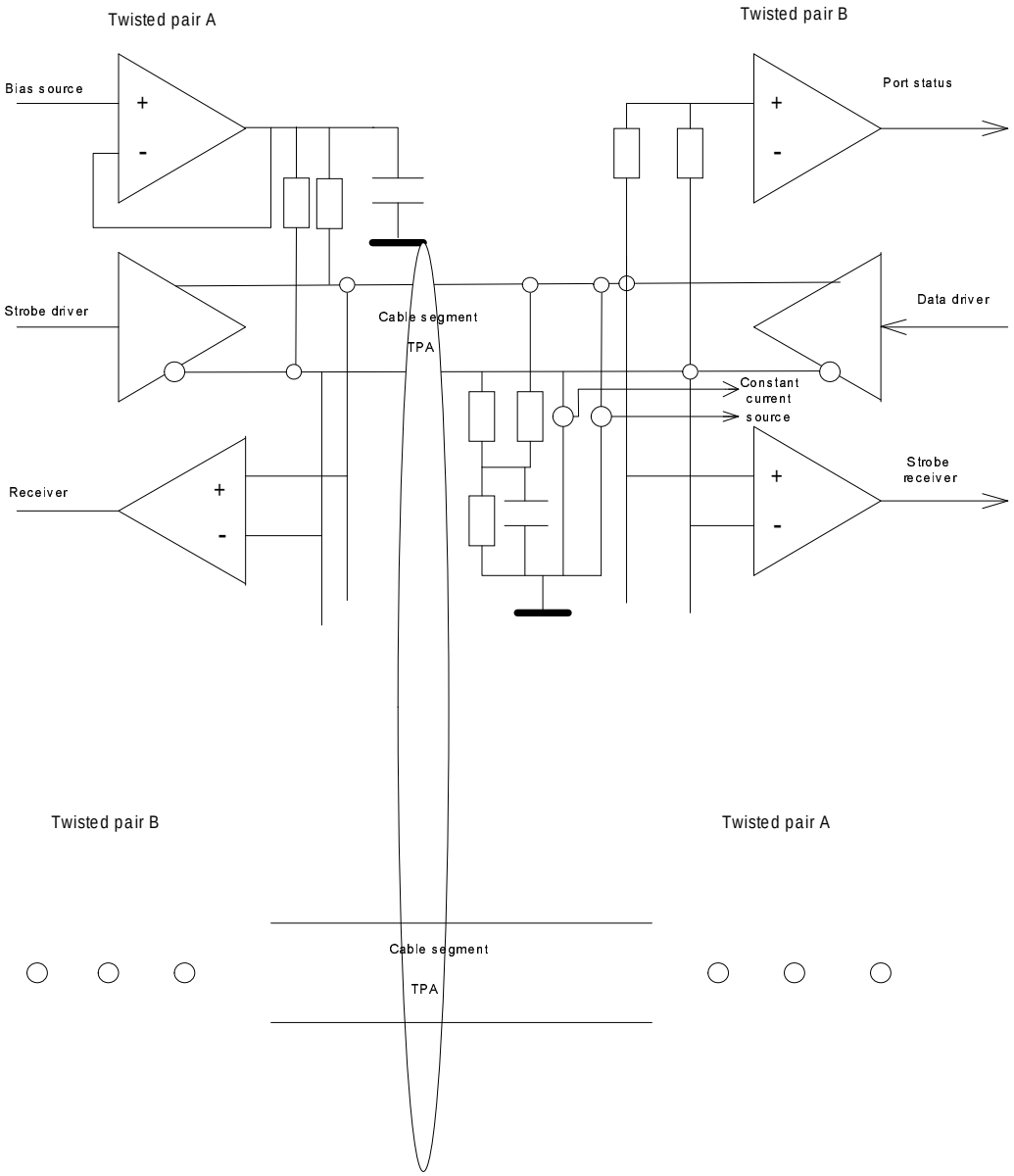
FireWire

Как сказано выше FireWire использует NRZ кодирование при котором число переходов между нулем и единицей меньше чем в NRZI, однако это компенсируется при помощи сигнала стробирования, передаваемого по отдельной паре проводов. Сигнал стробирования является дополнительным к сигналу данных в том смысле что CLOCK сигнал постоянной частоты может быть получен из этих двух сигналов при помощи побитового исключающего или.



Приемные цепи, так же как и в USB, принимают дифференциальные сигналы, ниже приведена схема входных и выходных цепей FireWire драйверов.

рисунок со схемотехникой


Twisted pair A
Twisted pair B
Bias source
Port status
Strobe driver
Data driver
Cable segment
TPA
Constant
current
source
Receiver
Strobe
receiver
Twisted pair B
Twisted pair A
Cable segment
TPA

Управление питанием

USB

Для классификации USB устройств по отношению к питающему напряжению вводится единица нагрузки по току величиной в 100 миллиампер (в зависимоти от условий питающее напряжение в разных частях шины может окахываться в диапазоне от 4.40 до 5.25 вольт).

Устройства, включая HUB, делятся на следующие категории:

1.HUB устройства питающиеся от шины (под этим для USB устройств понимается использование напряжения попадающего на устройство через порт, направленный в сторону хоста). Само устройство может потреблять одну единицу (см. выше опрделение) в течении всей работы шины, а так же еще 5 единиц, которые делятся между внутренними функциями и устройствами, подключенными к портам. При этом в каждый порт может отдаваться не более одной единицы. Сигналы могут проходить через HUB если он находится в активном или спящем состоянии. В силу ограниченности энергетических возможностей подобные HUB устройства обычно выполняют роль разветвителей в тех случаях, когда в компьютере не предусмотрено достаточного количества разъемов (встроенных USB шин или портов на встроенных HUB).

2. HUB устройства с собственым источником питания могут потреблять одну единицу для поддержания способности передавать сигнал в случае если собственный источник питания выключен. В спящем режиме HUB должен обеспечивать не менее 5 единый для подключенных к нему устройств. В состоянии активности каждый порт может получать не менн 5 единиц, однако ток, питания для одного порта не должен превышать 5 ампер. Питание от HUB может подаваться только в порты направленные в сторону от хоста. (так же как устройства с автономным питание не должны пытаться отдавать часть его остальной шине)
3. Устройства с питанием от шины и низким энергопотреблением, такие устройства потребляют не более одной единицы мощности и сохраняют работоспособность при понижении напряжения до минимального уровня (4.4 вольт)
4. Устройства с большим потреблением мощности потребляют в полностью включенном режиме до 5 единиц и до1 в режиме с низким потреблением. При подключении к шине такие устройства должны начинать работу в режиме с низким потреблением и так же как устройства предыдущего типа сохранять работоспособность при пониженном напряжении. ПО сможет включить устройство только если обнаружит что шина обладает достаточными энергетическими ресурсами.
5. Устройства с собственным источником питания могут потреблять так же не более одной единицы питания от шины если они находятся в состоянии с выключенным собственным источником.

Поскольку каждое устройство при инициализации сообщает хосту свои потребности в электроэнергии, все управление питанием устройств на шине распределяется между ПО хоста и функциями HUB устройств, которые имеют специальную функциональность для управления отключением портов. Аппаратное обеспечение самих компьютеров в свою очередь снабжено способностью выводить сам компьютер из спящего режима при обнаружении активности на шине USB.

FireWire

В отличие от USB, устройства FireWire могут не только иметь собственный источник питания, но и поставлять питающе напряжение через шину другим устройствам. Стандарт IEEE1394 определяет целую систему упреждающего оповещения о возможной потере тем или иным устройством способности поставлять питающее напряжение в шину.

Соединительные кабели могут как содержать пару проводов, предназначенную для передачи питающего напряжения, так и не содержать таковой. В силу этого, вся шина может разбиваться на отдельные домены, питание которых через шину обеспечивается различными устройствами или группами устройств.

С точки зрения питающего управления питанием устройства все устройства можно разделить на следующие категории:

1. Источники питающего напряжения
2. Альтернативные источники
3. Потребители
4. Устройства с собственным питанием, не относящиеся к остальным категориям

Следующая таблица дает расшифровку значений POWER_CLASS для устройств (POWER_CLASS передается устройствами в SelfID пакетах).

POWER_CLASS (двоичное)	***Описание***
0	Устройство не использует питание от шины и не передает его дальше
1	Устройство имеет собственное питание и поставляет в шину не менее 15 ват
10	Устройство имеет собственное питание и поставляет в шину не менее 30 ват
11	Устройство имеет собственное питание и поставляет в шину не менее 45 ват
100	Устройство может использовать до 3 вт мощности питания от шины, дополнительной мощности для поддержания активным уровня LINK не требуется (используется устройствами альтернативными источниками питания, а так же устройствами потребителями)
101	Зарезервировано для будущих реализаций
110	Устройство может использовать до 3 ват мощности питания от шины, для поддержания активным уровня LINK не требуется еще 3 ват
111	Устройство может использовать до 3 ват мощности питания от шины, для поддержания активным уровня LINK не требуется еще 7 ват

Устройства источники напряжения.

Устройства, являющиеся источниками питания, как основные, так и альтернативные должны иметь 6 контактные разъемы 1394 (включающие провода питания). Для того, чтобы в случае если напряжение на проводах питания в шине окажется выше напряжения выдаваемого данным устройством основным источником, устройство не поглощало энергию из шины используется диодная защита. Версия стандарта IEEE1394-1995 предусматривает один единственный диод, соединяющий внутренний источник питания устройства с портами, в отличие от этого IEEE1394a требует наличия отдельного диода для каждого порта. Таким образом 1394a предусматривает что основной источник разбивает шину на домены независимые по питанию – в случае если источник питания отключен,

диоды на портах не позволяют току проходить через устройство насквозь. Однако даже если подача источником питания в шину отключена, устройство может предавать насквозь сигналы передачиданных. Для этого, разумеется, должен быть включен уровень PHY, однако энергия для питания самого PHY чипа не должна браться из шины. Устройства, способные работать в таком режиме, заявляют POWER_CLASS равный 0.

Для IEEE1394a требования к выдаваемому питающему напряжению несколько отличаются от IEEE1394-1995. Максимальный ток для одного порта до 1.5 ампер, максимальнодопустимое напряжение 33 вольта, девиация напряжения в диапазоне частот от 1 килогерца до 400 мегагерц не должна превышать 100 милливольт. Для устройств имеющих POWER_CLASS 1,2 или 3 минимальное напряжение на выходе не менее 20 вольт, для остальных не менее 8 вольт.

Устройства альтернативные источники.

Альтернативные источники так же имеют 6 контактные разъемы портов (рекомендуется не менее двух портов). Подключение диодов соответствует подключению их в источниках IEEE1394-1995, таким образом альтернативные источники могут пропускать через себя питающе напряжение насквозь. В случае если собственная цепь подачи напряжения альтернативного источника отключена, для того чтобы сохранить способность способность передавать так же и сигналы насквозь, альтернативные источники могут поддерживать включенным собственный уровень PHY за счет внешнего питания, получаемого от шины. Используемый POWER_CLASS имеет значение 4.

Устройства потребители

Потребители очевидно имеют 6 контактные разъемы и POWER_CLASS равный 4,6 или 7. Для поддержания уровня PHY активным таким устройствам требуется не более 3 ватт, После получения PHY пакета link-on, такие устройства могут потреблять дополнительную энергию для работы уровня LINK. Если требования устройству к потреблению питания не укладываются полностью ни в один POWER_CLASS, то директории Configuration ROM содержат дополлнительную информацию.

Устройство с собственным источником питания

Если установлен POWER_CLASS равный 0, разъемы портов могут иметь как 4 так и 6 контактов, однако передавать насквозь питающее напряжение такие устройства не могут. Такое поведение соответствует так же устройствам источником с отключенным внутренним питанием.

Для POWER_CLASS равного 4, устройства безпрепядственно пропускают питающее напряжение насквозь в любую сторноу и так же как источники с POWER_CLASS равным 4 могут сохранять уровень PHY активным для сквозной передачи данных даже если уровень LINK и вышележащие уровни устройства отключены.

Спящий режим

Так же как и USB шина FireWire кроме местного отключения питания поддерживает режим suspend, на уровне отдельных портов. Выход из этого режима может происходить как программно при помощи посылки PHY пакета resume, так и по событиям подключения/отключения устройства на данном порту или изменения величины смещения. Способность пробуждаться от того или иного типа события зависит от информации, занесенной в регистры и может управляться программно.

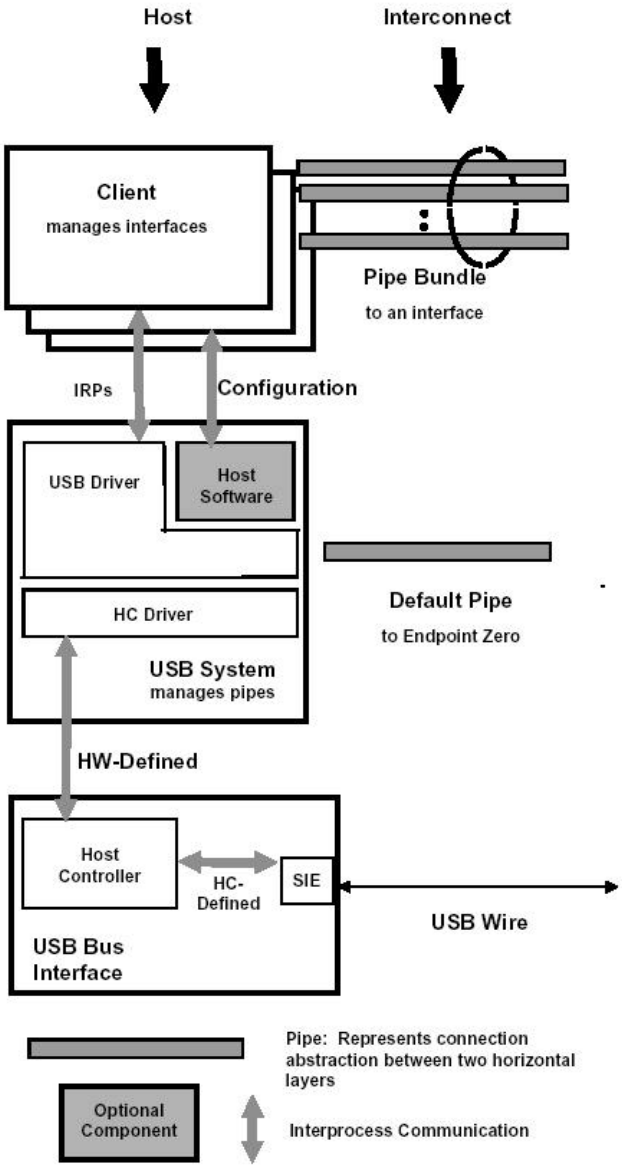
Работа программы (или драйвера) с шиной

USB

USB обеспечивает более высокий уровень протокола передачи данных, при написании программы (или драйвера) в большинстве случаев не следует заботиться о размере пакетов и порядке передачи данных.

Организуя потоковую передачу данных USB подсистема операционной системы берет большую часть этих забот на себя. Кроме того, в силу централизованности управления шиной со стороны хоста, при взаимодействии программного модуля с устройством возникает гораздо меньше неоднозначностей чем в случае FireWire. Шина USB компьютерно-ориентированная и поэтому воспринимается как несколько более удобная для программирования.

Рисунок (организация программной поддержки USB операционной системой компьютера)



Типичный API, предоставляемый операционными системами позволяет получить список всех подключенных устройств и узнать для каждого из них VendorID и ProductID и выбрать устройство для своей работы. Или во всяком случае (как например в Windows) программа, являющаяся драйвером может сообщить системе, что ее интересуют только устройства из списка возможных пар значений VendorID и ProductID. Какой бы способ ни был реализован, так или иначе наша программа сможет получить уведомление о подключении или отключении интересующего ее устройства.

Дальнейшее представляется достаточно простым. Программа должна знать протокол работы устройства. Большая часть информации об устройстве может быть получена и большая часть настроек могут быть выполнены при помощи default endpoint номер 0 выбранного устройства.

Каждое устройство может иметь несколько конфигураций и информация о них так же может быть получена от устройства. В каждый момент времени активной является только одна из конфигураций.

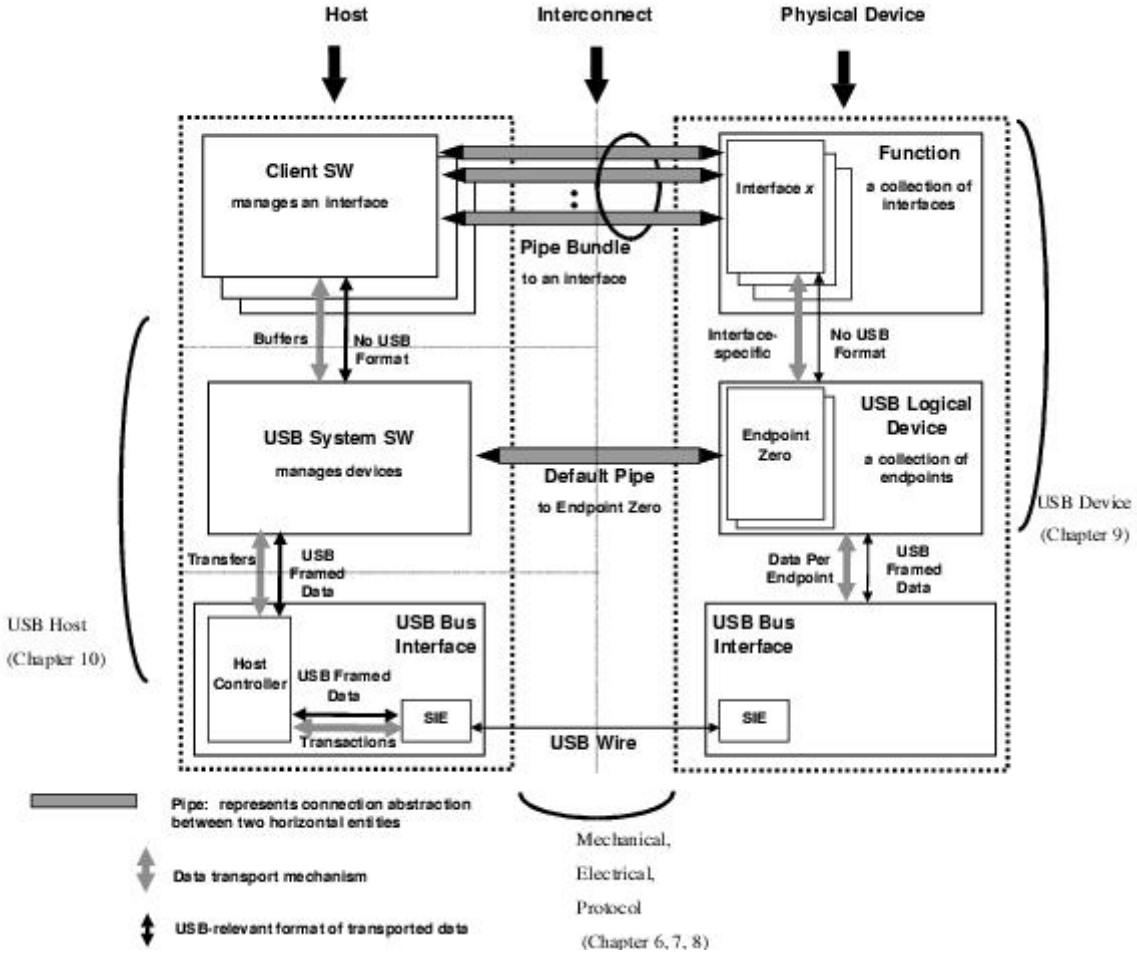
Некоторые из устройств, подключенных к шине могут являться устройствами типа HUB и API позволяет программам обойдя все дерево устройств узнать точную топологию шины.

Каналы передачи isochron, async, interrupt с точки зрения языка программирования представляются потоками ввода или вывода. Пожалуй основным отличием каналов interrupt является то, что если прерывание не пришло в течении некоторого интервала времени поток, может перейти в состояние stalled.

Как правило программы или драйверы владеют устройством монопольно и используют каналы, предоставляемые устройствами по своему усмотрению.

Рисунок (Каналы передачи и уровни реализации протокола USB)

Доступ к каналам может осуществляться как через обычные потоки, так и через



функции ioctl. Данные для каналов типа control как уже говорилось должны передаваться в виде пакетов определенного формата, кроме того каналы control являются двунаправленными. Так или или иначе данные для каждого канала всегда передаются и принимаются последовательно. Поэтому с точки зрения программирования USB предоставляет более удобный интерфейс чем FireWire, где вся процедура усложняется в силу того, что требуется производить запись и чтение массивов данных по разным адресам виртуального адресного пространства. При этом результат в принципе может зависеть например от того, будет ли тот или иной участок памяти прочитан (или записан) за одну операцию чтения/записи или за несколько. Последнее требует достаточно строгого соблюдения протокола работы с самой шиной и с устройством. Для USB похожая ситуация может иметь мемто только для control в силу того, что пакеты должны иметь правильный размер и структуру. В случае посылки неправильных данных при работе с control ошибка может наступить по истечении таймаута в качестве значения которого спецификация определяет значение 5 секунд. Однако API для данной ОС (в частности) может позволять указывать другое значение этого таймаута для каждой конкретной операции работы с каналом типа control.

Множество медленных устройств (low speed, 1.5Mbit/sec) использует каналы control (и в частности default pipe 0) в качестве канала для изохронной передачи данных в одну или даже в обе стороны. Таковыми,например, являются электронный ключ Aladdin Etoken и устройство чтения записи чиповых карт ACR30U, которые упоминались во введении. Напомним, что low speed устройства могут использовать только interrupt и isochron каналы!

Ридер смарткарт ACR30U получает от хоста команды через канал типа control и возвращает через канал типа interrupt.

EToken – это криптографическое устройство, поддерживающее различные алгоритмы шифрования и электронной подписи (DES,RSA,DSA...) и представляет из себе объединенный в одном корпусе ридер и смарткарту. Кроме того, EToken может иметь внутри себя до 64 килобайт памяти, представленной файловой системой. Для передачи команд и ответов на команды используется исключительно канал default pipe 0 типа control. Поверх этого канала реализован протокол команд яиповых карт ISO7816.

Исходные тексты программ и драйверов для работы с этими устройствами можно найти на сайте www.linuxnet.com.

Более подробно с работой чиповых карт и протоколом ISO7816 можно ознакомиться в книге [3], [6] .

Стоит так же упомянуть о Web-камере Logitech QuickCam, о которой так же говорилось во введении. Камеры этого типа не используют изохронную передачу, как это обычно делается когда необходимо передавать видео в реальном времени, вместо этого используется передача типа bulk. Очевидно что на фактическую скорость передачи изображения может влиять текущая загруженность шины. Напомним что при использовании изохронного канала передача изображения не сможет начаться если шина не сможет выделить для этого канала необходимую полосу пропускания! Драйвер для ОС Linux этой камеры так же доступен в Интернет в виде исходных текстов на сайте SourceForge.

Использование изохронной передачи в настоящее время более характерно для устройств IEEE1394 чем для USB в силу того что шина FireWire изначально предназначалась для передачи видео. Как видно каждая шина имеет свою характерную специфику и поэтому вопреки ожиданиям экспертов USB и FireWire так и не стали конкурирующими стандартами. Шина USB наиболее дешевая и технологичная для применения с компьютером, в отличие от нее FireWire, будучи более сложной и дорогой, обладает разветвленной интеллектуальностью. Вероятно именно поэтому характерно что обе шины встроенные в в компьютер в настоящее время встречаются восновном в достаточно дорогих нотбуках, которые одновременно являются как бы и носимыми устройстваи как и видеокамеры Sony, стоимость которых так же довольно высока. Тем не менее FireWire имеет в этом случае и чисто компьютерное применение, такое как работа с внешними дисками и TCP/IP сеть

поверх протоколов IEEE1394.

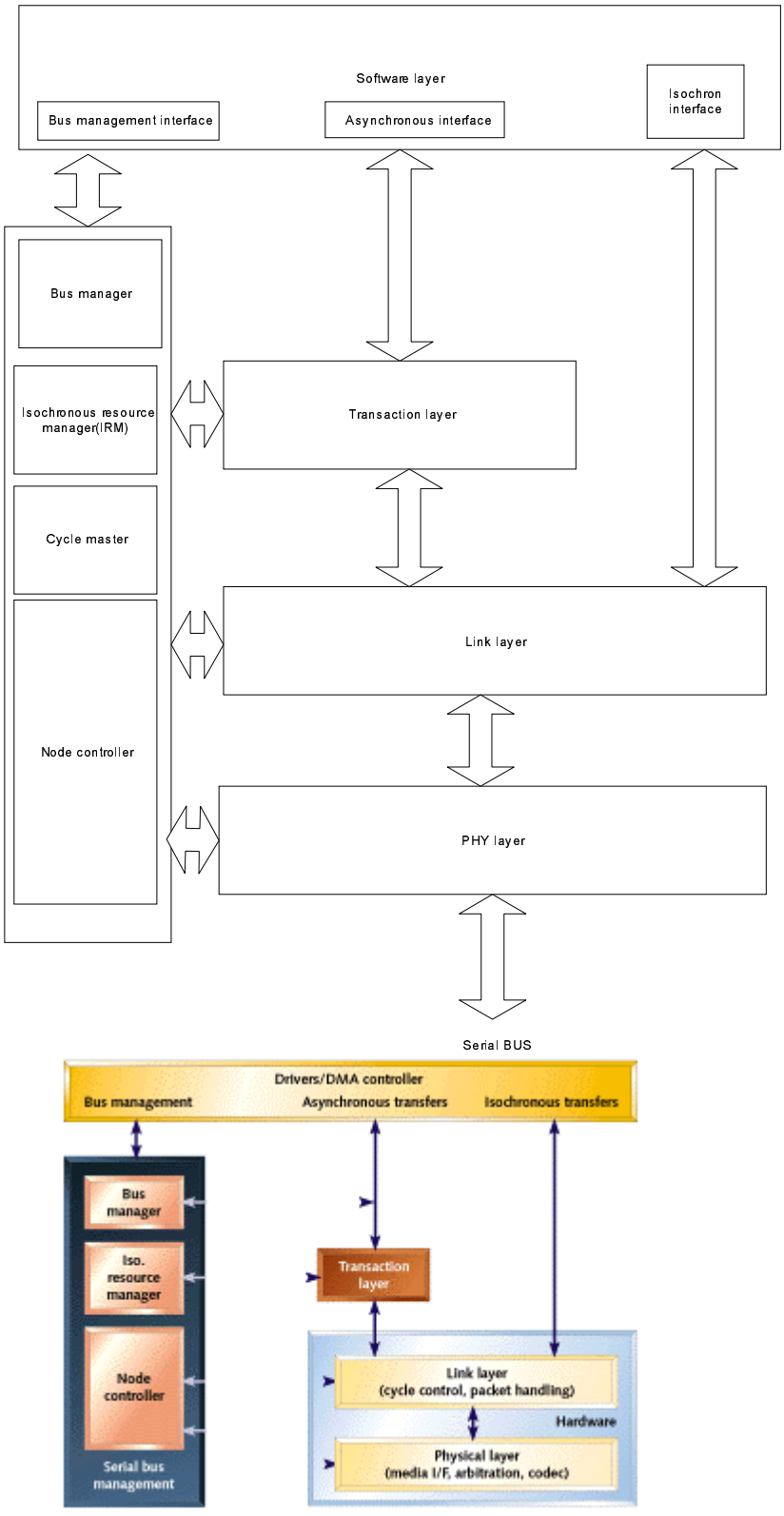
Существует еще один достатлчно широкий класс медленных USB устройств. Это так называемые HID устройства (HID расшифровывается как Human Interface Device). HID – это протокол, реализованный поверх USB, основной целью которго является формализация работы компьютера с устройствами ввода типа мышей джойстиков и.т.п., этот протокол имеет дело уже не с каналами предачи а с кнопками, расположенными на внешних устройствах. ОС Windows позволяет обращаться к HID устройствам не только драйверам, но и пользовательским программам, по\тому HID устройства легко могут быть использованы в качестве “тренажера” по использованию USB. В качестве более полной возможности ознакомится с работой USB можно рекомендовать API для Java в ОС Linux. 3 Существует два таких API (javax.usb и jUSB), каждое из которых позволяет полностью использовать все возможности шины по работе с устройствами. Подробную информацию о них можно найти в Интернете на сайтах SourceForge и java.sun.com.

FireWire

Взаимодействие ПО с шиной происходит как на уровне транзакций (Transaction layer) так и на уровне связи (Link layer) для чего определены понятия программных интерфейсов FireWire: интерфейс асинхронных транзакций, интерфейс изохронной передачи, минующий уровень транзакций и обращающийся напрямую к Link layer, а так же интерфейс управления шиной (Bus management interface). Каждый из этих программных интерфейсов как правило обеспечивается отдельным драйверов ядре ОС. Эти интерфейсы используют прикладные драйверы устройств, например таких как внешний накопитель FireWire.

Написание собственных драйверов устройств для Windows может оказаться задачей не из легких, а кроме того ошибки допущенные при написании драйвера с легкостью могут привести к порче данных на диске компьютера и к подвисанию системы. Поэтому во всяком случае для того, чтобы начать писать программы, работающие напрямую с устройствами FireWire разумно использовать специальные системные библиотеки, позволяющие пользовательскому процессу, находящимуся вне ядра ОС использовать соответствующие API. Одной из операционных систем, предоставляющих такую возможность является Linux, где с помощью системной библиотеки libraw1394 можно без риска нарушить целостность ядра во время работы использовать все доступные функции шины FireWire.

Рисунок (Уровни протокола FireWire)



Фактичекси работа программы с устройствами сводится к следующему:

1.получение через интерфейс управления шиной необходимой информации об устройствах (тип устройств, максимальная скорость передачи, топология подключения к шине, нахождение менеджера изохронных ресурсов, проверка наличия достаточных по мощности источников питания и.т.п.)

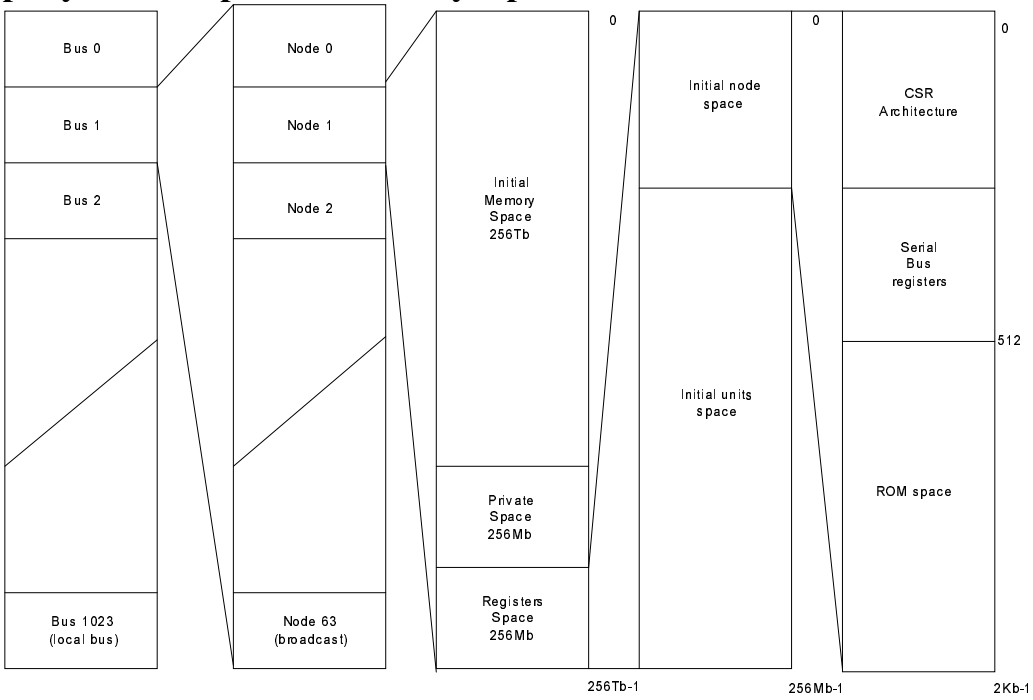
2. через интерфейс управления шиной выделяются каналы и настраиваются для изохронной передачи, предоставляются сервисы доступные другим устройствам и выполняются

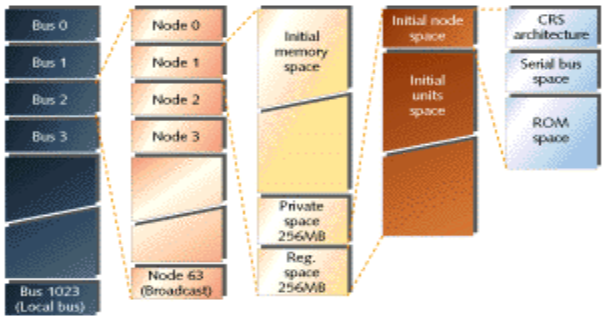
прочие действия по настройке шины.

3. происходит обмен данными с выбранным устройством при помощи асинхронных операций чтения/записи по виртуальным адресам, находящимся в устройстве, а так же получение и прием данных через выделенные ранее изохронные каналы.

Каждое устройство представлено областью памяти 256Тб в конце которой по адресу 0xFFFF'F000'0000 находится область занимающая 256 мегабайт и называемая CSR-регистрами, при помощи операций асинхронного чтения записи в эти регистры происходит вся процедура управления каждым из устройств. Структура CSR определена специальным стандартом ISO/IEC-13213, нас будет интересовать группа регистров, являющихся обязательными для FireWire устройств. Стандарт IEEE1394 определяет лишь часть этих регистров.

рисунок “карты памяти” устройства





Мы собираемся сначала описать назначение и местонахождение наиболее важных регистров, а затем рассмотреть на примере возможный сценарий их использования.

Начиная со смещения в 1 килобайт от адреса 0xFFFF'F000'0000 располагается Configuraton ROM, которая содержит информацию об устройстве. На основе этой информации ОС может определить какой драйвер следует использовать для данного устройства, может ли устройство выполнять функции bus manager и.т.д. Для Configuration

ROM определена минимальная структура, которая содержит только идентификатор производителя (Vendor ID) и структура общего типа, которая в свою очередь содержит различную информацию, специфичную для данного устройства.

Основные регистры

Адрес(а) относительно 0xFFFF'F000'0000	***Название***	***Является обязательным***	***Описание***
0x0	STATE_CLEAR	да	Состояние и управляющая информация (обязательно реализован)
0x4	STATE_SET	да	Установка бита STATE_CLEAR (обязательно реализован)
0x8	NODE_IDS	да	Задает 16 битовый NodeID (обязательно реализован)
0xC	RESET_START	да	Восстанавливает исходное состояние узла
0x10-0x14	INDIRECT_ADDRESS INDIRECT_DATA	нет	Косвенный доступ к ROM>1Кб
0x18-0x1C	SPLIT_TIMEOUT	Зависит от устройства	Таймаут Split-request (если split транзакции поддерживаются то обязательно реализован)
0x20-0x2C	TUSARGUMENT, TEST_START, TEST_STATUS	нет	Диагностический интерфейс
0x30-0x4C	UNITS_BEASE, UNITS_BOUND, MEMORY_BASE, MEMORY_BOUND	нет	Не используется
0x50-0x54	INTEARRUPT_TARGET, INTERRUPT_MASK	нет	Broadcast/nodecast interrupt

Адрес(а) относительно 0xFFFF'F000'0000	*Название*	*Является обязательным*	*Описание*
0x58-0x7C	CLOCK_VALUE, CLOCK_PICK_PERIOD, CLOCK_STROBE_ARRIVED, CLOCK_INFO	нет	Синхронизированное время
0x80-0xFC	MESSAGE_REQUEST, MESSAGE_RESPONSE	нет	Регистр для передачи сообщений
0x100-0x17C	RESERVED	нет	Зарезервировано для CSR
0x180-0x1FC	ERROR_LOG_BUFFER	нет	Зарезервировано для шины
0x200-0x3FC	SERIAL_BUS_DEPENDANT	да	Регистры, специфичные для шины (bus dependant), расшифровывается в следующей таблице (обязательно реализованы)

Основные регистры
Регистры, специфичные для шины (bus dependant)

Адрес(а) относительно 0xFFFF'F000'0000	*название*	*описание*
0x200	CYCLE_TIME	Используется узлами, поддерживающими изохронную передачу для поддержания синхронизации cycle time
0x204	BUS_TIME	Узлы, способные выполнять роль cycle master для всей шины используют этот регистр для текущего значения bus time
0x208	POWER_FAIL_IMMENENT	Запись в режиме broadcast в этот регистр оповещает о скором возможном отключении питани

Адрес(а) относительно 0xFFFF'F000'0000	*название*	*описание*
0x20C	POWER_SOURCE	Используется вместе с POWER_FAIL_IMMENENT для процедуры подтверждения возможного отключения питания
0x210	BUSY_TIMEOUT	Используется узлами, поддерживающими transaction retry
0x214-0x217	Не исп.	Зарезервировано
0x218	FAIRNESS_BUDGET	Определяет количество разрешенных дополнительных попыток захватить шину для асинхронной передачи в течении fairness interval
0x21C	BUS_MANAGER_ID	Адрес узла, выполняющего роль bus manager
0x220	BANDWIDTH_AVAILABLE	в менеджере изохронных ресурсов используется для учета свободной полосы пропускания в единицах 1/4915 от 100 микросекундного интервала.
0x224-0x228	CHANNELS_AVAILABLE	в менеджере изохронных ресурсов используется для учета свободных изохронных каналов
0x22C	MAINT_CONTROL	Для диагностических целей
0x230	MAINT_UTILITY	Для отладочных целей
0x234-0x3FC	Не используется	Зарезервировано

Мы собираемся кратко описать назначение основных регистров.

BUS_TIME и CYCLE_TIME

Устройства, использующие изохронную передачу обязаны иметь эти регистры. Содержимое регисттров обновляется с частотой 24.576МГц, отчитывая 125 микросекундные интервалы (тайм слоты).
CYCLE_TIME

7 бит second_count	13 бит cycle_count	12 бит cycle_offset

BUS_TIME

25 бит second_count	7 бит second_count_lo

cycle_offset отсчитывает 125 микросекундные интервалы по модулю 3071

cycle_count отсчитывает по модулю 7999 и по обнулению переносит 1 в регистр second_count , который в свою очередь отсчитывает односекундные интервалы.

second_count в свою очередь считает до 127 и увеличивает second_count_hi при этом second_count всегда совпадает с second_count_lo.

POWER_FAIL_IMMENENT и POWER_SOURCE

Регистр POWER_FAIL_IMMEENT является опциональным, но если он реализован в устройстве, то должен быть обязательно реализован и POWER_SOURCE. В случае если некоторое устройство, подающее питание в шину обнаруживает, что ему придется прекратить подачу напряжения, оно может произвести запись в регистр POWER_FAIL_IMMENENT других устройств на шине при помощи broadcast адреса 0x3F. Получившие такой сигнал устройства могут сравнить адрес устройства пославшего оповещение с адресом, содержащимся в POWER_SOURCE и если они совпадают, то принять возможные меры, связанные с отключением питания.

При записи в POWER_FAIL_IMMENENT используется следующий формат: установленный первый бит(pri_flag) говорит о валидности пакета, следующие 15 бит (pri_delay) определяет время в сотнях миллисекунд до наступления возможного отключения питания, последние 16 бит (pri_source) содержат полный адрес устройства источника оповещения (10 бит номер шины и 6 бит адрес устройства). Последние 16 бит регистра POWER_SOURCE так же содержат адрес источника питания.

MAINT_CONTROL MAINT_UTILITY

Если реализованы, то служат для диагностических целей, например позволяют проверку правильности передачи пакета или заставить устройство вернуть тот или иной статус ошибки для следующей транзации.

BUS_MANAGER_ID

Этот регистр всегда реализован в устройствах, которые способны выполнять роль менеджера шины (bus manager). Устройства могут пытаться присвоить себе роль bus manager и для этого таким устройствам потребуется модифицировать содержимое регистров BUS_MANAGER_ID, для достижения транзакционности для этой цели используются транзакции дипа lock. Младшие (из 32) 6 бит этого регистра могут содержать идентификатор (адрес) текущего менеджера шины. Изначально эти биты инициализируются значением 0x3F, что является broadcast адресом. При посылке пакета по адресу 0x3F, пакет будет получен всеми устройствами.

BANDWIDTH_AVAILABLE и CHANNELS_AVAILABLE

Регистр CHANNELS_AVAILABLE состоит из двух октетов (всего 64 бтита), каждый установленный бит отвечает за свободный изохронный канал.

Регистр BANDWIDTH_AVAILABLE имеет следующую структуру:

19 бит (зарезервировано)	13 бит (значения от 0 до 4915) свободная полоса пропускания

Поскольку несколько устройств могут одновременно захотеть выделить изохронный канал для своих целей необходим механизм, обеспечивающий транзакционность при конкурировании за ресурсы каналов и полосы пропускания. Для реализации такой транзакционности служит асинхроная команда записи Lock. В своем варианте Lock compare эта команда позволяет произвести запись в область памяти как атомарное действие при условии, что текущее состояние области памяти соответствует некоторому состоянию, которое используется в качестве параметра для команды. Команда Lock завершается успешно в случае совпадения значений и ошибочно в случае несовпадения.

Для того, чтобы создать изохронный канал устройство должно найти менеджер изохронных ресурсов, прочитать его регистры CHANNELS_AVAILABLE , BANDWIDTH_AVAILABLE, вычислить какие изменения нужно внести в эти регистры с тем, чтобы получить канал нужной скорости передачи и попытаться изменить эти значения при помощи Lock (если, конечно, будет обнаружено что для этого свободно достаточно ресурсов). В случае если один регистр был успешно изменен, а при изменении второг возникла ошибка, следует произвести в первом регистре обратные изменения. В случае успеха – следует уведомить другое устройство о необходимости начать передавать данные и сообщить ему номер изохронного канала. Последнее делается способом, специфическим для каждого конкретного устройства.

Устройство, выделившее изохронный канал отвечает за повторное выделение этого же канала в случае перезагрузки шины (bus reset) вызванной подключением нового устройства (когда содержимое CSR регистров теряется). Для того, чтобы все, существовавшие до перезагрузки каналы могли восстановиться и устройства смогли продолжить передачу, устройства не имевшие ранее ч выделенных изохронных каналов (но желающие их зарезервировать) должны выждать после перезагрузки некоторое время прежде чем резервировать новые каналы.

Асинхронная посылка данных происходит при помощивызова команд read/write/lock соответствующего API. Получение же данных возможно при помощи loop-back интерфейса, когда приложение регистрирует в нижележащих драйверах специальную функцию, которая будет автоматически вызываться драйвером каждый раз когда другое устройство захочет произвести асинхронную запись по виртуальным адресам, находящимся в нашем устройстве (в роли которого в данном случае выступает компьютер). В эту функцию драйвер передает всю необходимую информацию, касающуюся запроса, касающуюся запроса от другого устройства, включая адрес и длину записываемых или считывающих данных.

Для асинхронной передачи имеет место такая же в точности ситуация, за исключением того, что вместо адреса там используется номер логического изохронного канала. При изохронной передаче программа должна своевременно принять все пришедшие данные а так же своевременно отослать данные в устройство. Для этого существуют интерфейсы для оповещения программы о том, что данный изохронный канал в зависимости от напраления передачи по нему либо требует добавления данных для отправки либо наоборот данные получил и необходимо срочно эти данные прочитать стем чтобы освободить внутренний буфер драйвера, связанный с этим каналом.

Что произойдет если программа не успеет вовремя принять или послать блок изохронных данных зависит от конкретной реализации внешнего устройства, некоторые устройства могут “относиться спокойно” к подобной ситуации, совершенно не обязательно что устройство перейдет в состояние ошибки, вовремя недополучив например кадр видеоизображения! Возможно просто изображение будет воспроизведено с некоторыми

дефектами. Фактически изохронный канал гарантирует, что шина в каждом таймслоте выделит достаточно времени для передачи данных со скоростью заявленной для этого канала, однако устройства могут использовать предоставленную им полос пропускания и не полностью. Из описанмя схемы арбитража шины, которое приведено ранее, понятно что при этом неиспользованное время может быть использовано асинхронными транзакциями.

Unit registers

Начиная со смещения 0x800 от начала Initial Units Registers располагаются регистры Unit registers, наиболее важными из которых являются
TOPOLOGY_MAP (диапазон адресов 0x1000-0x13FC)
SPEED_MAP (диапазон адресов 0x2000 – 0x2FFC)
Устройство, которое оказывается менеджером шины, запоминает первые 4 байта из SelfID пакетов, в том порядке в котором их посылают устройства во время процедуры Self Identification.
На основании этой информации любое устойство, которому это потребуется может восстановить топологию шины.
Формат TOPOLOGY_MAP следующий:
16 бит длина всей таблицы.
16 бит CRC.
32 бит generation_number – количество раз, которое менеджер шины генерировал карту топологии шины.
16 бит число подключенных устройств
16 бит self_id_count – общее число SelfID пакетов, посланных устройствами.
Дальше сдедуют 4 байтовые фрагменты SelfID пакетов в количестве self_id_count.

Регистр SPEED_MAP содержит от 0 до 4029 чисел, определяющих максимальную скорость. передачи между каждой парой устройств. Заголовок такой же в регистре TOPOLOGY_MAP, в след за которым размещены однобайтовые элементы (симметричной) матрицы скоростей.

PHY Registers и Configuration ROM.

Регистры PHY обеспечивают низкоуровневый интерфейс к шине, они могут быть прочитаны другими устройствами, желающими получит информацию о числе портов, максимальной скорости передачи, версии поддерживаемого стандарта IEEE1394 и.т.п., поддерживаемых данным устройством. Некоторые из этих регистров могут быть изменены при помощи посылки пакетов PHY Configuration. Например можно дать устройству комнаду задержать свое участие в процедуре Tree Identification на 167 микросекунд, с тем чобы оно не смогло получить роль менеджера шины.

Кроме того PHY регистры содержат информацию VendorID и ProductID, позволяющие определить тип устройства и иденцифицировать драйвер, который необходимо использовать для него. В следующей таблице приводится полный список регистров версии стандарта 1394a (1394-1995 имеет существенные отличия, соответствующие данные приводятся в конце этого раздела), функциональность некоторых из них подробно мы не обсуждаем. Байт со смещением 0x110 зарезервирован и не используется, пары регистров (Extended и Total_ports, Max_speed и Repeater Delay, Page_select и Port_select) разделены внутри своих байтов одним неиспользуемым битом. Для каждого регистра начиннающегося на границе байта в таблице приведено смещение этого байта.

РегистрPHY IEEE1394a	*Запись/чтение*	*Длина (bit)*	*Описание*
Physical ID	R	6	В процессе Self identification сюда заносится адрес устройства (смещение 0000)
R	R	1	В процессе Self identification сюда заносится признак того, что устройство является корневым узлом.
PS	R	1	Power status – уровень PHY устанавливает этот бит если обнаружено подходящее питающее напряжение на шине 7.5-33 вольт.
RHB	R/W	1	Root Hold Off – может быть установлен специальным конфигурационным PHY пакетом и заставляет устройство задержать свое участие в конкурировании за роль корневого узла. Назначение этой возможности обсуждалось в главе про Tree identification. (смещение 0001)
IBR	R/W	1	Initiate Bus reset – установка этого бита начинает процесс перезагрузки шины, через 166 микросекунд бит автоматически сбрасывается.
Gap_count	R/W	6	По умолчанию после bus reset содержит значение 63. Интервал Gap используется для оптимизации передачи по шине при разных временах задержки в кабеле. Узлы bus manager и isochronous manager могут изменить значение, записанное в этом регистре.
Extended	R	3	Значение 7 в этом регистре указывает на то, что используется расширенный формат регистров.(смещение 0010)
Total_ports	R	4	Количество портов, реализованных в данном узле.
Max_speed	R	3	Максимальная скорость, поддерживаемая узлом.(смещение 0011)
Repeater Delay	R	4	Худшее время задержки репитора, выражаемая как 144+(delay*20)ns.
Link active	R/W	1	Может быть программно переписан, PHY использует этот регистр для broadcast(смещение 0100)
Contend	R/W	1	Contender – может быть изменен программно, очищается при перезагрузке шины.
Repeater Delay Jitter	R	3	Разница времен максимального и минимального времени задержки репитора, вычисляемая по формуле: (jitter+1)*20ns
Power Class	R/W	3	Это значение передается в SelfID пакете, указывает тип устройства по отношению к получению и подаче питания на шине.
Resume_int	R/W	1	При установке этог бита узел обязан установить Port_event бит, если порт находился в состоянии suspend и вывести порт из этого состояния(смещение 0101)

РегистрPHY IEEE1394a	*Запись/чтение*	*Длина (bit)*	*Описание*
ISBR	R/W	1	Initiate Short Bus Reset – при устоновке вызывает так называемую короткую перезагрузку шины, (использующую механизм арбитража). После перезагрузки автоматически сбрасывается.
Loop	R/WC	1	Получает значение 1 в случае обнаружения петель в топологии шины, обнаруживаемых по истечению таймаута в процесе Tree Identification. Может быть сброшен записью 1.
Power Fail	R/WC	1	Получает значение 1 если питающее напряжение шины падает ниже 7.5 вольт. Может быть сброшен записью 1.
Timeout	R/WC	1	Указывает на истечение таймаута автомата арбитража (200-400 микросекунд).Может быть сброшен записью 1.
Port_event	R/W	1	Устанавливается аппаратно при обнаружении изменения величины электрического смещения порта, подсоединения, перехода в состояние disabled или fault в случае если установлен бит Int_Enabled. Также устанавливается в случае resume если установлен бит Resume_int.
Enab_accel	R/W	1	Разрешает ускоренный арбитраж если это расширение поддерживается.
Enab_multi	R/W	1	Разрешает конкатенацию пакетов разных скоростей.
Page_select	R/W	3	Выбирает регистровую страницу 000 страница порта, указанного в Port_select 001 страница VendorID 111 страница Vendor dependent (смещение 0111)
Port_select	R/W	4	Указывает к какому порту будет относиться страница выбранная Page_select

Страница порта содержит информацию относящуюся к состоянию выбранного (при помощи регистра Port_select) порта устройства.

Регистр страницы порта	*Длина (bit)*	*Чтение/запись*	*Описание*
AStat	2		Состояние выходов пары сигнальных проводов TPA 11=”Z”,01=”1”,10=”0”,00=”Invalid”.(смещение 1000)
BStat	2		Аналогично Astat, но для TPB.
Ch	1		1–Child port, 0-Parent port.
Con	1		1 – порт подсоединен, 0 – порт свободен

Регистр страницы порта	*Длина (bit)*	*Чтение/запись*	*Описание*
Bias	1		Установлен в 1 если входные цепи обнаруживают электрическое смещение.
Disabled	1		Установлен в 1 если порт запрещен
Negotiated speed	3		Максимальная скорость для данного порта, кодируется так же как и содержимое регистра Max_speed.(смещение 1001)
Int_Enable	1		Разрешает регистр Port_event если установлено в 1 (реакция на события Connected, Bias, Disabled, Fault)
Fault	1		

Страница VendorID содержит следующую информацию:

1. Со смещением 1000 -битовый Compiance_level уровень поддерживаемого стандарта (0 – не указан, 1 – 1394a, остальные значения для следующих версий).
2. Со смещением 1010 находится 24 битовое значение VendorID.
3. Со смещением 1101 так же 24 битовое значение ProductID

Формат и содержание восьми регистров страницы Vendor dependent определяется производителем.

Для PHY регистров установлен страничный режим адресации, в зависимости от значения занесенного в регистр Page_select область сосмещениями 0x1000-0x1111,следующая сразу за Port_select, представляет одну из трех восьмирегистровых страниц: Port Status Page, Vendor Identification Page, Vendor-dependant information page. Каждый регистр на этих страницах имеет длину в один байт.

В заключении приводим аналогичные данные для IEEE1394-1995:

РегистрPHY IEEE1394-1995	*Длина (bit)*	*Описание*
Physical ID	6	В процесе Self identification сюда заносится адрес устройства
R	1	В процесе Self identification сюда заносится признак того, что устройство является корневым узлом.
PS	1	Power status – уровень PHY устаноавливает этот бит если обнаружено подходящее питающее напряжение на шине 7.5-33 вольт. (смещение 0000)
RHB	1	Root Hold Off – может быть установлен специальным конфигурационным PHY пакетом и заставляет устройство задержать свое участие в конкурировании за роль корневого узла. Назначение этой возможности обсуждалось в главе про Tree identification. (смещение 0001)
IBR	1	Initiate Bus reset – установка этого бита начинает процесс перезагрузки шины, через 166 микросекунд бит автоматически сбрасывается.

РегистрPHY IEEE1394-1995	*Длина (bit)*	*Описание*
Gap_count	6	По умолчанию после bus reset содержит значение 63. Интервал Gap используется для оптимизации передачи по шине при разных временах задержки в кабеле. Узлы bus manager и isochronous manager могут изменить значение, записанное в этом регистре.
SPD	2	Определяет максимальную скорость передачи, поддерживаемую уровнем PHY: 00 – 100Mb/sec 01 – 200Mb/sec 10 – 400Mb/sec 11 – зарезервировано (смещение 0010)
E	1	Enhanced bit – если установлен, то означает что реализованы регистры использующие пространство после #Ports+0100
#Ports	5	Число имеющихся в узле портов, соответственно число следующих далее регистров статуса будет то же самое
AStat[n]	2	Содержит состояние выходов для витой пары TPA: 11 - “Z” 01 - “1” 10 - “0” 00 – invalid (Astat[0] имеет смещение 0100)
BStat[n]	2	Содержит состояние выходов для витой пары TPB: 11 - “Z” 01 - “1” 10 - “0” 00 - invalid
Ch[n]	1	Установлен в 1 если к порту подсоединен узел “child” и в 0 если “parent”
Con[n]	1	Установлен если к порту подключен другой узел
ENV	2	Используется вместе с “enhanced register” (см. поле E), имеет следующие значения: 00 – backplane 01 – cable 10 и 11 – зарезервированы (смещение 0100)

РегистрPHY IEEE1394-1995	***Длина (bit)***	***Описание***
Reg_count	6	Указывает количество дополнительных регистров (см. поле E), которые следуют непосредственно далее начиная со смещения 0101

Configuration ROM

Для регистров Configuration ROM предусмотрен минимальный формат, который содержит только 24 битовый VendorID и общий формат, представляющий собой набор директорий, содержащих специфическую информацию.
Минимальный формат имеет следующий вид:

8bit 0x01	24bit VendorID

Формат общего вида содержит заголовок, корневую директорию и поддиректории. Пример того, как правильно прочитать и расшифровать содержимое Configuration ROM, можно найти в документации по видеокамере Sony, которая обсуждается ниже в этой главе.

Кроме того, устройство может быть составным и содержать в себе несколько единиц (unit) в этом случае Configuration ROM, содержит отдельные директории с информацией о каждой единице.

Структура ROM общего формата состоит из ряда блоков (включая директории), длина которых кратна октету (32 бита). Заголовок этой структуры состоит из следующих полей: info_length (8 бит), crc_length (8 бит), rom_crc (16 бит). Поле info_length позволяет программам отличать ROM минимального формата от ROM общего формата. Для минимального формата первые 8 бит содержат число 1. В свою очередь crc_length указывает общую длину памяти, покрытой CRC.

Вслед за заголовком находится состоящий из четырех октетов bus_info_block, Первый его октет содержит ASCII представление названия шины (“1394”). Второй октет содержит информацию о способностях устройства по отношению ка работе шины, для 1394a формат является расширением такового для 1394-1995:

Регистр	***Длина (бит)***	
irmc	1	Флаг способности устройства быть менеджером изохронных ресурсов
cmc	1	Флаг способности выполнять роль Cycle master
isc	1	Флаг поддержки изохронной передачи
bmc	1	Флаг способности быть менеджером шины
pmc	1	Флаг способности выполнять роль Power manager (1394a)
reserved	4	
cyc_clk_acc	8	Для устройств с установленным cmc содержит значение точности тактов часов в единицах миллионных долей секунды (допустимые значения от 0 до двоичного 100), для прочих устройств заполнен единицами.
max_rec	4	Задает значение максимальной длины асинхронного пакета с которым может работать данное устройство. Длина вычисляется как 2 в степени единица плюс значение данного регистра. В настоящее время используются значения от 0 до 10,
reserved	4	
gen	2	Флаг, указывающий изменялось ли содержимое Configuration ROM со времени предыдущего bus reset(1394a)
reserved	3	
link_speed	3	(1394a)

И наконец последние два октета, содержат node_vendor_id 24 бит и вслед за ним 40 бит chip_id. Эти 64 бита должны однозначно идентифицировать данное устройство. То же самое значение содержится в директориях в структуре Node_unique_id leaf.

Следом за bus_info_block располагается структура директорий, начинающаяся с Root_directory. В корневой директории ROM общего формата должны обязательно присутствовать следующие поддиректории: Module_Vendor_Id, Node_Capabilities, Node_Unique_Id. Директория Node_Unique_Id в версии стандарта 1394a более не является обязательной так как составляющие Node_Vendor_ID и Chip_ID доступны в bus_info_block.

Директории и файлы имеют следующий формат: 2 бита тип ключа, 6 бит значение ключа, 24 бита содержимое. Тип ключа определяет как же собственно найти объект, возможны следующие варианты: 0 означает что далее следует непосредственно значение, 1 указывает на то, что далее идет значение смещения (от начала Initial Registers Space) где находится параметр, 2 означает что далее идет ссылка на файл и наконец 3 означает ссылку на директорию. Значение ключа определяет назначение и тип объекта, возможные значения приведены в таблице.

Значение ключа	
0x03	Module_Vendor_Id
0x0C	Node_Capabilities
0x0D	Node_Unique_Id
0x02	Bus_Dependant_Info (директория или файл)
0x04	Module_Hardware_Version
0x05	Module_Spec_Id
0x06	Module_Software_Version
0x07	Module_Dependent_Info (директория или файл)
0x08	Node_Vendor_Id
0x09	Node_Hardware_Version
0x0A	Node_Spec_Id
0x0B	Node_Software_Version
0x10	Node_Dependent_Info (директория или файл)
0x11	Unit_Directory (одна для каждого входящего в состав unit-a)

Длполноты изложения в таблице ниже приводятся так же битовые флажки Node_Capabilities (каждый флажок имеет длину 1 бит и первый флажок начинается в позиции 9, т.е. Первые 8 бит не используются), их назначение здесь не обсуждается:

Флажок в Node_Capabilities	***Описание***
spt	Поддержка Split timeout
ms	Реализованы регистры Messaging_passing
int	Реализованы регистры Interrupt_target и Interrupt_mask
ext	Реализованы регистры Argument
bas	Реализованы регистры Test_start и Test_status

Флажок в Node_Capabilities	***Описание***
prv	Данный Node реализует private space
_64	Используется 64 разрядная адресация
fix	Используется фиксированная адресация (всегда установлен)
lst	Реализован Lost_bit в State_clear и State_set
drq	Реализован Disable_request в State_clear и State_set
r	Зарезервирован
elo	Реализован регистр Error_log и соответствующий бит в в State_clear и State_set
atn	Реализован бит Attention в State_clear и State_set
off	Реализован бит Off в State_clear и State_set
ded	Поддерживается режим dead state
init	Поддерживается режим initializing state

Структура Node_Unique_Id следующая: соответствующий файл содержит в своем заголовке смещение, указывающее на содержимое состоящее из 16 битового значения 0x0002, CRC16 бит (покрывающее оставшиеся поля), 24 ,bn Node_vendor_id, 40 бит Chip_id.

Из описания регистров CSR (Control and Setup Registers) видно, что реализация всего этого набора со стороны устройства может оказаться не таким уж простым делом по сравнению с с USB и даже PCI устройством. Задачу может несколько упростить наличие стандарта на реализации чипов FireWire уровней PHY и LINK, называемого OHCI-1394 (Open Host Controller Interface). Тем не менее с этой задачей в настоящее время весьма успешно справляются некоторые отечественный производители электронного оборудования, такие как например “Инструментальные Системы” (www.insys.ru). “Инструментальные Системы”предлагают специализированные системы сбора данных, некоторые из которых используют FireWire для связи с компьютером.

Рассмотрим как все это может использоваться на примере цифровых видеокамер Sony XCD-SX900 и XCD-X700, подробное описание протоколы работы которых можно найти на сайте производителя в файлах GXCDSX900E.pdf и GDFWV500E.pdf.

Для того, чтобы видеокамера начала передачу изображения необходимо определить какое из подключенных к шине устройств является камерой и проинициализировать ее после этого камере необходимо сообщить желаемые настройки: частоту кадров и формат передачи изображения. Камере так же необходимо указать какую из двух доступных скоростей передачи следует использовать (200Mbit/sec или 400Mbit/sec). Разумеется выбранная скорость передачи должна быть достаточна для передачи требуемого числа кадров в секунду в выбранном формате. После этого необходимо создать изохронный канал нужной пропускной способности, (более подробно процедура описана ранее), и сообщить его номер камере с тем чтобы она начала передачу. После этого остается в цикле принимать пакеты с изображением!

Обычно процедура создания канала начинается с определения адреса устройства менеджера изохронных ресурсов. Этот адрес можно прочесть в регистрах устройства менеджера шины. Обычно API предоставляет возможность узнать эти адреса. Для создания

канала необходимо прочесть и модифицировать при помощи функции lock регистры BANDWIDTH_AVAILABLE и CHANNELS_AVAILABLE в менеджере изохронных ресурсов с тем чтобы часть свободной полосы отвести под вновь создаваемый канал.

Возможно все это кажется несколько сложным, однако если не считать расчета величины на которую следует изменить BANDWIDTH_AVAILABLE, все остальное сводится к вызову несколько раз функций read/write/lock.

Для того чтобы “обнаружить” камеру нужно получить при помощи API список всех устройств и прочитав VendorID и ProductID из Configuration ROM каждого из них запомнить адрес устройства найденной камеры. Если единственным подключенным устройством является камера, эту процедуру можно пропустить. Дальнейшее, согласно описанию протокола приводимому производителем, сводится примерно к следующей последовательности команд:

1. Записать в регистры камеры по адресу 0xFFFF'F0F0'0000 значение 0x8000'0000 и тем самым проинициализировать камеру.
2. Выбрать количество кадров изображения в секунду (например 7.5) записав по адресу 0xFFFF'F0F0'0600 значение 0x4000'0000
3. Выбрать разрешение картинки, скажем 1024x768 (запись по адресу 0xFFFF'F0F0'0604 значения 0xA000'0000)
4. Установить допустимый для выбранного разрешения формат передачи 0xFFFF'F0F0'0608 значение 0x2000'0000
5. Сообщить камере номер созданного ранее изохронного канала и скорость передачи (запись по адресу 0xFFFF'F0F0'060C значение 0xN1000'0000 или 0xN2000'0000, в зависимости от выбранной скорости, где N – номер канала от 0x0 до 0xF)
6. И наконец включить передачу изображения записав по адресу 0xFFFF'F0F0'0614 значение 0x8000'0000

В принципе ничего сложного! Во всяком случае не сложнее передачи через последовательный порт RS232 с использованием программного управления потоком :)

Разумеется на практике для того, чтобы при передаче изображения не происходило его потерь, нам потребуется приемный буффер достаточно болшого размера, обработка ошибочных ситуаций и тому подобное, но суть дела это не меняет. Теперь мы представляем себе как работать с FireWire устройствами. Подсистема FireWire нашей ОС при приеме данных от устройства через изохронный канал просто будет вызывать нашу функцию и передавать в нее пришедшие данные, наша забота теперь только успевать их обрабатывать!

Изохронная передача работает примерно так же (примеры можно найти в исходных текстах ОС Linux), однако все может быть достаточно гладко до тех пор пока принимающее устройство является достаточно интеллектуальным чтобы правильно использовать принимаемый поток данных на своей стороне. Речь идет во о чем, например при получении изображения если это изображение требуется показывать на экране принимающий компьютер может определять какой кадр видео и когда когда следует начать и закончить показывать чтобы изображение не дергалось и не замирало, вобщем вело себя адекватно. Однако в качестве принимающего устройства могут попасться существенно менее интеллектуальные камкордеры, которые показывают каждый кадр изображения ровно тогда когда ими этот кадр получен. И что же в результате? Камкордер, использующий телевизионный формат NTSC для внутреннего представления изображения для правильной работы должен получать 29.97 видеокадров в секунду. Изображение передается внутри пакетов протокола CIP (Common Isochronous Packets). Размер кадров составляет 480x250, соответственно один кадр занимает 120000 байт. Таким образом NTSC требует скорости передачи 29.97*120000=3596400 байт в секунду. Изохронные пакеты IEEE1394 передаются с частотой 8КГц и если выбрать для канала наиболее близкую к требуемой скорость передачи, то с учетом всех полей заголовка CIP пакета окажется что один такой пакет будет передавать 480 байт изображения. Но для того, чтобы достичь скорости передачи 29.97 кадров в секунду потребовалось бы в одном пакете передавать 3596400/8000=449.55 байт в секунду! Простые устройства не могут преобразовывать потоки видео с одной частотой кадров в потоки видео

с другой частотой. Но к счастью CIP допускает другое решение этой проблемы: примерно 1 из 15 CIP пакетов должен содержать внутри себя указание что он не несет в себе байтов данных изображения, такие пакеты игнорируются устройством. Более подробную информацию о передаче изображения можно найти в документации и исходных текстах Linux.

Аппаратные решения для устройств USB и FireWire

USB

Производители электронных компонентов выпускают специализированные микросхемы, реализующие протоколы USB хоста и устройства. С использованием этих микросхем можно реализовать собственные аппаратные решения для шины USB. Кроме того, для реализации каких-либо дополнительных возможностей не предусмотренных производителями можно воспользоваться микросхемами программируемой логики (PLIS), в качеств отправной точки для этого можно воспользоваться например VHDL прошивками USB драйверов доступными на сайте www.opencores.org. Последнее может быть интересно так же с точки зрения изучения электрического протокола шины.

Для того, чтобы быстро реализовать собственное устройство с USB интерфейсом проще воспользоваться одной из серийно выпускаемых микросхем. В качестве примера можно привести микросхему USBN9604 производства National Semiconductor, полное описание этой микросхемы свободно доступно через Интернет. С помощью USBN9604 можно достаточно быстро изготовить USB устройство “собственного производства” (много полезных советов на эту тему можно найти в конференции www.telesys.ru и на сайте http://www.is.svitonline.com/vks/
).

Поскольку требуется регистрация собственных VendorID и ProductID для промышленного выпуска USB устройств производители микросхемы зарегистрировали таковые для нее, в документации сказано что эти VendorID и ProductID разработчики устройств при соблюдении некоторых условий могут использовать по своему усмотрению. Понятно, что использование одинаковых значений различными устройствами может затруднить написание драйверов для них и даже сделать драйвера для устройств от различных поставщиков недопустимыми для одновременной установки в системе. Однако такой подход вполне может быть выходом для производства мелкосерийных специализированных устройств, кроме того, если с устройствами должен работать не драйвер а напрямую пользовательская программа (как это возможно например в ОС Linux), неприятностей практически вообще не должно быть.

FireWire

Аппаратные интерфейсы шины IEEE1394 в отличие от USB не разделяются на хосты и устройства, однако существуют комплекты микросхем предназначенные для сложных полнофункциональных устройств, таких как например компьютеры, и для более узкоспециализированных, например видеокамер.
Чипы FireWire более жестко стандартизированы (OHCI 1394) чем таковые для USB, при большей сложности логики работы шины 1394 это вполне оправданно. В частности OHCI определяет электрический интерфейс между микросхемой FireWire контроллера шины и самим устройством, его использующим. Например в зависимости от максимальной скорости передачи 100,200 или 400 мегабит в секунду в чипах, реализующих уровень PHY со стороны самого устройства используются используются 1,2 или 3 провода.

Тем не менее существуют реализации чипов не соответствующие или не совсемсответствующие OHCI. Реализации, предназначенные для хоста имеют интерфейс к шине PCI, и различия в реализациях выливаются в различия драйверов, поддерживающих работу с FireWire контроллером как с PCI устройством. На сегодняшний день существуют следующие основные реализации хост-контроллеров:
1.OHCI-1394 совместимые контроллеры

2.Adaptec AIC-5800
3.Texas Instruments PCILynx

Уровни PHY и LINK четко разделены в OHCI и интерфйсы между ними так же строго специфицированы. FireWire позволяет управлять активностью PHY и LINK поотдельности, в частности при помощи механизма конфигурационных PHY пакетов.

В качестве примера микросхем, реализующих уровни LINK и PHY можно провести чипы производства Texas Instruments (более подробная информация на сайте www.ti.com):
Рекомендованные для хостов с шиной PCI это TSB12LV22 (LINK) и TSB41LV03 (PHY 400Mbit/sec) и для устройств типа видеокамер TSB12LV31 GPLynx (LINK) и TSB21LV03 (PHY 200Mbit/sec)

Спецификация OHCI является открытой и может быть бесплатно получена через Интернет.

Поддержка FireWire, USB различными платформами

Ни смотря на то, что в настоящее время USB устройства являются достсточно распространенными, поддержка их различными ОС несколько отставала от успехов производителей соответствующего “железа”. Так, например, если возможность работать напрямую с собственным устройством, подключенным к последовательному порту RS232 при помощи стандартного компонента из программы Visual Basic или Delphi является весьма обычным делом, то это вряд ли можно сказать про USB, а тем более FireWire.

Существуют два осовных способа работы прикладной программы с USB/FireWire устройствами: первый - обращение к специализированным API драйверов верхнего уровня, которые обеспечивают доступ к абстрактному устройству (сканеру, принтеру, диску и.т.п.), скрывая при этом от программы детали, связанные с работой последовательной шины и второй способ, когда используется API нижнего уровня, предоставляющее доступ непосредственно к шине на уровне пакетов, каналов передачи, регистров CSR и.т.п.

Не все из ныне существующих ОС поддерживают второй способ, для таких систем приходится выбирать второй, означающий что для реализации “заветной мечты” электронщика увидеть как происходит передача данных из его устройства на микроконтроллере Atmel и USB чипе в настольный PC, ему придется писать специализированный драйвер и загружать его в ядро ОС! Неприятности этого пути очевидны:

1. требуется хорошее знание ядра и навыки написания драйверов
2. в случае ошибки можно не только “подвесить” систему, но и испортить содержимое диска
3. сложно вносить много мелких изменений, которые обычно необходимы при разработке собственного “железа”

Пожалуй единственной ОС, принципиально лишенной этих недостатков при использовании второго подхода, можно считать QNX [6]. В современных реализациях QNX драйверы работают в пространстве прикладных программ (user space). Если драйверы выполняются с правами суперпользователя, то им будут доступны все необходимые ресурсы. Вообще следует отметить, что QNX по-видимому обладает наиболее гибкой системой реализации драйверов по сравнению с другим распространенными ОС.

Но что делать если мы пока только лишь мечтаем об использовании подобных сверхгибких систем, что нам предлагают другие ОС, к которым мы уже привыкли, рассмотрим вкратце другие варианты:

Windows

ОС Windows как раз относится к тем системам в которых доступ пользовательских программ к USB/FireWire весьма ограничен. Пожалуй исключением является возможность использовать USB устройства типа HID. Прикладная программа, однако, может получить список всех устройств, подключенных к USB шине.

Возможность работы с HID все же позволяет организовать тестирование работы USB устройства “собственного производства”, однако устройству придется реализовывать работу с протоколом HID (который будучи предназначенным для мышей, джойстиков, клавиатур может оказаться не совсем адекватным) и довольствоваться низкими скоростями передачи. Но все же это лучше чем ничего!

Другой альтернативой является написание собственного драйвера, в Интернет можно найти исходные тексты какого-либо драйвера USB устройства и приспособить его под свои задачи. Для этого понадобится Visual Studio .NET, а так же Microsoft SDK и DDK (Driver Developer's Kit). Пакет SDK можно бесплатно скачать сайта Microsoft, что же касается

DDK, то компакт диск содержащий его, Microsoft за небольшую плату высылает по почте всем желающим. Следует отметить что SDK должен соответствовать версии компилятора, а DDK должен так же соответствовать версии ОС. Для каждой из версий Windows98/2000/NT/XP придется делать свой вариант драйвера. Взятый за основу драйвер USB/FireWire устройства так же должен соответствовать всем перечисленным компонентам. Кроме того, потребуется еще соответствующая документация по kernel API. Если вы ставите перед собой задачу статьразработчиком драйверов для Windows – это безусловно ваш путь, но если вы занимаетесь разработкой устройств и пока не владеет всем этим, вам наверное следует выбрать другие пути. Можно начать с HID устройств, а можно еще попробовать другие ОС.

Исходя из всего вышесказанного естественным образом возникает следующая мысль: а нельзя ли сделать такой драйвер, чтобы он предоставлял прикладным программам возможность использовать USB или FireWire по своему усмотрению. Такие попытки существуют с разной степенью успешности и можно попытаться найти некоторые реализации драйверов в Интернет http://www.steelbrothers.ch/jusb/. Трудность состоит в том, что в Win32 драйверы строго привязаны кVendorID и ProductID конкретного устройства. Поэтому наш специализированный драйвер должен будет при попытке прикладной программы создать в системе такую привязку. Последнее восновном сводится к занесению записи в системный реестр. Если нам интересно использовать драйвер для тестовых целей – эту привязку можно в принципе выполнить заранее
вручную! К сожалению не все версии Windows позволяют сделать это без перезагрузки системы. Последнее так же может затруднить отладку подобного драйвера. И все же при большом желании это возможно, хотя вряд ли может быть рекомендовано для того, чтобы начать самостоятельно работать с USB устройствами. На SourceForge готовится к выходу версия реализации javax.usb API (JSR80, которая по-видимому тем или иным способом решит эту проблему http://cvs.sourceforge.net/viewcvs.py/javax-usb/javax-usb-ri-windows/ ,
javax.usb API создана восновном стараниями корпорации IBM, как одного из самых активных сторонников Java Community Process.

Linux

ОС Linux предлагает полный набор возможностей для использования доступа к USB и FireWire из прикладной программы. Кроме того, в исходных текстах системы и драйверов можно найти достаточное количество примеров с некоторым количеством комментариев.

Доступ программы к USB возможен при помощи так называемой USB File System (основные функции определены в файлах usb.h, usbdevice_fs.h) при помощи IOCTL. В качестве примера можно например использовать программы для работы с USB ридером ACR или устройством Etoken (которые обсуждались выше), доступные на www.linuxnet.com. В качестве простого примера использования каждой USB функции можно рассматривать исходные тексты нативной библиотеки для jUSB (одного из известных API для Java, о котором пойдет речь чуть позднее).

Структуры IOCTL для работы пользовательской программы определены в файле */usr/include/linux/usbdeviceb_fs.h*, соответственно константы в файле */usr/include/linux/usb.h*. За информацией об использовании USB интерфейса Linux из драйверов отсылаем читателя к соответствующим руководствам по написанию модулей ядра. Файл *usbdeviceb_fs.h* определяет множество структур, каждая из которых отвечает за IOCTL для соответствующей операции с шиной, на первый взгляд это может показаться достаточно сложно, в частности например для того, чтобы организовать потоковую передачу например через канал типа BULK или ISOCHRON, Кроме того, перед началом работы с устройством требуется выполнить еще несколько специальных IOCTL команд для того, чтобы затребовать соответствующий USB интерфейс. Для удобства имеет смысл скрыть все эти особенности

внутри специализированных оберточных классов на C++, Objective C или другом объектноориентированном языке. Все это довольно несложно сделать основываясь на вышеназванном примере из jUSB, к тому же проект jUSB предлагает готовый C++ интерфейс, дублирующий аналогичный интерфейс для Java.

Использование FireWire в Linux весьма аналогично использованию USB. Специальная библиотека Libraw1394 предназначена для использования пользовательской программой, данная библиотека существенно упрощает использование команд IOCTL, предоставляя набор специализированных функций. Для использования этих функций нужно иметь достаточное представление о работе самой шины IEEE1394, мы надеемся что прочтение сего текста позволило читателю получить всю необходимую для этого информацию ;) Кроме того, к библиотеке прилагается описание ее API (которое так же должно быть понятно "посвященному в секреты FireWire") и несколько достаточно понятных примеров программ http://www.linux1394.org/ , кроме того данная страница содержит краткое, но достаточно полезное описание самого стандарта FireWire.

Для доступа к USB и FireWire пользовательской программе потребуются права суперпользователя root. Если это ограничение покажется слишком обременительным, возможно переделать драйвер так, чтобы файлы, отвечающие за соответствующие устройства создавались с другими правами доступа нежели как доступные только root.

Во введении упоминалась WEB-камера Pyro, это устройство вполне можно выбрать для первых экспериментов с драйверами FireWire устройств для Linux, дело в том, что работа с видеокамерой существенно проще чем с блочными устройствами типа внешних накопителей. Драйвер Linux для этого устройства достаточно простой для понимания, недостатком является более высокая цена камеры по сравнению с WEB-камерами USB. С другой стороны Pyro является устройством более высокого класса чем большинство распространенных WEB-камер. Если в распоряжении имеются 2 компьютера с FireWire адаптерами, то можно попробовать и другие варианты: например FireWire TCP/IP сеть или даже просто запись/чтение через шину регистров при помощи одного компьютера в другом компьютере. Последнее впринципе достаточно несложно осуществить пользуясь библиотекой libraw1394.

Платформа Java.

Java в своих реализациях под существующие ОС использует нижележащее API системы, предоставляя лишь удобный хорошо документированный объектноориентированный интерфейс программирования. Поэтому все ограничения ОС распространяются на соответствующие Java API. Тем не менее эти API могут быть использованы в качестве удобного средства, позволяющего сконцентрироваться на взаимодействии с устройством а не на особенностях программирования в данной платформе, для работы с USB устройствами в Linux.

В настоящее время существуют два стандартных API: GNU jUSB и JSR80 javax.usb. http://jusb.sourceforge.net/
http://sourceforge.net/projects/javax-usb/

В Linux доступ к USB устройствам осуществляется как к специальным файлам, , а поскольку при подсоединении эти файлы устройств возникают автоматически, необходимо чтобы текущий пользователь получил права на уровне разделения доступа, самый простой способ этого достичь – это использовать привелегии суперпользователя root для процесса, открывающего файл или же после подсоединения устройства изменить условия доступа к нему вручную. Пакет jUSB предлагает еще одно решение этой проблемы – запуск с правами root специализированной программы-демона JUSBD, тогда прикладная программа, соединившись по сети с JUSBD сможет получить доступ к USB, Кроме того эта прикладная

программа может выполняться на другом компьютере, в частности под Windows.

SUN Solaris

В ОС Solaris ситуация напоминает таковую в Windows, за тем лишь исключением, что исходные тексты ядра ОС могут быть доступны программисту, что может несколько облегчить задачу.

Исходные тексты программ и другие источники информации в интернет

В этом небольшом разделе мы собираемся перечислить некоторые полезные Интернет ресурсы, не претендуя между тем на полноту охвата всевозможных источников информации.

На сайте, посвященном встраиваемым системам читатель может найти краткое введение в основы протокола IEEE1394 http://www.embedded.com/1999/9906/9906feat2.htm.

Несколько полезных ссылок и исходных текстов программ можно найти на сайте посвященном поддержке FireWire в операционной системе Linux http://www.linux1394.org/.

Ряд различных публикаций, касающихся FireWire (IEEE1394), которые вероятно заинтересуют читателя доступны на по следующему адресу http://grouper.ieee.org/groups/1394/1/Documents.

Интернет страница международного USB консорциума содержит первоисточники документов, касающихся данного стандарт www.usb.org в отличие от IEEE1394 спецификация USB является абсолютно безплатной и именно она послужила основой для нашего изложения.

Стоит отметить ресурс http://www.linuxnet.com, который содержит ряд исходных текстов программ и драйверов для ридеров смарткарт http://www.linuxnet.com/ с интерфейсом USB, данный сайт может быть полезен как UNIX так Windows программистам, поскольку содержит множество “документации на языке С”, общем для всех операционных систем.

Примером хорошо написанного драйвера USB устройства может служить таковой для WEB-камеры Logitech QuickCam, исходные тексты которого находятся на http://qce-ga.sourceforge.net.

Для тех кого в первую очередь интересует как сделать собственное USB устройство будет интересен следующий русскоязычный сайт: ht
http://www.is.svitonline.com/vks/

Разнообразная информация, касающаяся IEEE1394, USB переодически появляется на сайте http://www.embedded.com.

Интернет страницы, касающиеся поддержки USB платформой Java находятся по следующим адресам: Java USB API http://www.steelbrothers.ch/jusb/ и javax.usb http://cvs.sourceforge.net/viewcvs.py/javax-usb.

Небольшая статься, посвященная сравнению USB и FireWire, находится в Интернет http://www.mycableshop.com/usbvsieee.htm. Эта статья возможно более полно раскроет различия между двумя обсуждавшимися здесь стандартами чем данная публикация.

Следует отметить что приведенные ссылки относятся в основном к англоязычным ресурсам не только в силу привычки автора получать самую последнюю информацию из последних первоисточников, но и в силу объективного отставания отечественных источников информации в данной области, этот пробел в какой-то мере и хотелось бы заполнить. Некоторое время назад столкнувшись с необходимостью получения фундаментальной информации по FireWire, было обнаружено практически полное отсутствие доступной русскоязычной литературы по этой теме! Впоследствии именно это побудило к написанию публикации, включившей в себя не только вновь приобретенные знания, касающиеся IEEE1394, но и некоторый опыт в использовании USB. Надеемся что изложение на основе сопоставления двух стандартов поможет читателю более полно понять принципы работы обеих шин. Изложение информации направлено на создание представления об обсуждаемых стандартах в целом, а так же на выяснение вопросов, которые могут возникнуть при работе с USB и FireWire. Несмотря на собственное пристрастие автора к OpenSource технологиям, ОС Linux и Java, все же была сделана попытка изложить основные структурные особенности не привязываясь к особенностям конкретной платформы.

В заключении мы собираемся привести краткий обзор некоторых других последовательных шин и беспроводных протоколов связи для локального соединения компьютеров.

Другие последовательные шины и беспроводные каналы связи.

Предшествующие реализации.

Стандарты получившие свое развитие в виде реализаций последовательных шин, получающих все более широкое распространение в настоящее время, имеют своими прародителями ряд стандартов прошлого поколения, а так же некоторые частные реализации разработанные отдельными компаниями и организациями. Вот некоторые из них:

1. Apple desctop bus (ADB) – последовательная шина реализующая протокол чтения записи для вплоть до 16 устройств подключенных к компьютеру Mac. Максимальная скорость передачи до 90 килобит в секунду.
2. Access.bus (A.b)– шина созданная Access.bus Industry Group, эта шина, созданная на основе шины I2C (Philips) так же как и USB предназначена для динамического подключения и отключения до 127 устройств. Шина использует арбитраж на электрическом уровне и позволяет передавать даные со скоростями до 100 килобит в секунду. Однако по сравнению с I2C шина A.b, оставаясь достаточно частной реализацией, в настоящее время большого распространения не получила.
3. Concentration Highway Interface (HCI) – шина, предложенная AT&T для передачи звука. HCI имеет централизованный источник тактирования и позволяет передавать данные со скоростями до 4 мегабит в секунду.
4. GeoPort – предлложен Apple Computing, предназначен для соединения двух устройств и позволяет осуществлять передачу со скоростями до 2 мегабит в секунду. Использует схему передачи сигналов RS422.

Как предшественники USB и FireWire следует упомянуть протоколы семейства RS, начиная со всем известного RS232. Среди этих протоколов есть такие, которые позволяют управлять сразу несколькими устройствами на растояния нескольких метров и даже десятков метров, эти разновидности в настоящее время достаточно широко используются в промышленности. Все эти шины объединяет наличие большого количества используемых проводов (порядка одного-двух десятков), отсутствие четкого стандарта на поведение устройств при их динамическом подключении/отключении, отсутствие механизма астоматического определения типа подключенных устройств.

Перечисленные недостатки устраняют шины следующего поколения, краткому обзору которых и посвящена эта глава.

Современные последовательные проводные шины

Шина I2C.

Среди распространенных последовательных шин I2C повидимому имеет наибольшее число различных (иногда несовместимых реализаций). Шина I2C имеет арбитраж на уровне электрических сигналов, каждая передаваемый блок данных включает в заголовок номер устройства, которому данный пакет адресован. Адреса устройств должны быть известны заранее. Существует множество вариантов шин близких по устройству к I2C, некоторые из них используют для передачи сигналов 2 провода, некоторые 3, некоторые из

них передают сигнал тактирования, некоторые “прячут” его в самих данных. Для некоторых примитивных устройств бывает даже необходимо передавать дополнительное число сигналов тактирования, зависящее от длины переданных данных. I2C и ее нестандартные разновидности нашли широкое применение в специализированных электронных устройствах, применением наиболее близким к персональным компьютерам и их периферийным устройствам пожалуй являются чиповые карты. В частности устройство ACR30, которое упоминалось среди прочих разновидностей карт, поддерживает работу с I2C картами и некоторыми картами, использующими нестандартные разновидности I2C. Как парвило – это наиболее простые карты памяти, содержащие несколько десятков байт данных или телефонные карты. Более подробно с чиповыми картами можно ознакомится например в книге “Смарткарты, настольная книга разработчика” [3]. Для чиповых карт более высокой ценовой категории обычно используется электрический протокол передачи, определенный стандартом ISO7816.

Шина CAN.

Абревиатура CAN расшифровывается как Cable Automation Network. Назначение CAN – управление оборудованием локомотивов, кораблей, других курпных подвижных объектов. Каждое устройство CAN имеет назначенный ему адрес. Шина так же как и I2C имеет механизм арбитража и позволяет посылать короткие команды и получать ответы от устройств. Характерной особенностью CAN, является способность передачи на расстояния в десятки и сотни метров а так же высокая устойчивость к электрическим наводкам. Так же как и в USB/FireWire в CAN применяется разностная передача сигналов. Существуют специальные репиторы, позволяющие еще больше увеличить расстояния доступные для передачи. В настоящее время распространение получает шина CAN2, имеющая повышенную прпускную способность и призванная удвлетворить растущие потребности в быстрой передаче все больших объемов данных. Все CAN устройства в рамках одной сети как правило подключаются к общей четырехпроводной шине.

Особый интерес CAN вызывает еще и потому, что не только производители больших трейлеров, но и компактных легковых автомобилей (такие как Mersedes, BMW, Ford) начинают внедрять эту технологию. На некоторых моделях (в частности BMW выпуска начиная с 1995 года) CAN является практически единственным каналом управления фарами, замками, стеклоподъемниками! Все чаще перед автосервисами встает вопрос необходимости освоения комплексного ремонта и обслуживания электрооборудовния таких автомобилей.

К сожалению автоприозводители не спешат раскрывать свои частные команды управления, реализованные поверх CAN. Однако коммерчески доступны (хотя и достаточно дороги) некоторые тулкиты третьих производителей для диагностики автомобильных CAN сетей. Кроме того, доступно специальное оборудование и свободное программное обеспечение CanOpen для самостоятельного подключения к этим сетям. Заинтересованного читателя отсылаем к интернет-сайту московской фирмы “Марафон” www.marathon.ru, can.marathon.ru.

Беспроводные сети и каналы передачи

Беспроводные каналы передачи с некоторой точки зрения можно так же классифицировать как последовательные шины, которые в качестве единственного провода используют эфир. Однако обычные методы разделения канала связи, используемые в прводных средах для передачи через эыир просто неприменимы.

Очевидно использования способов арбитража, используемых USB, FireWire,

CAN для эфира непрменимы, если только не пытаться использовать для арбитража отдельный частотный канал. Дело еще осложняетсятем, что обычно беспроводные каналы LAN (Local area network) и PAN (personal area network, например BlueTooth) используют так называемые нелицензируемые SIM (Scientific-industrial-medical) частоты. Обязательным требованием к устройствам, использующим эти частоты, является применение либо широкого спектра либо применения прыгающей частоты (Frequancy hopping). Способ избежания коллизий, применяемый в проводных сетях на основе Frame 802.2 и 802.3, таких как Ethernet так же неприменим в силу того, что устройство в момент собственной передачи не может слушать канал чтобы узнать не пытается ли другое устройство вести передачу одновременно с ним. Во всяком случае это потребовало бы для работы нескольких устройств набора достаточно сильно разнесенных по частоте каналов, каждый из которых использовался бы для передачи отдельным устройством.

Еще одной важной особенностью использования эфира по сравнению с проводной передачей является что типичной является следующая ситуация: устройства A и B каждое находятся в зоне видимости C, но друг друга видеть не могут. При этом в силу того, что устройство A не может определить момент начала передачи устройством B, оно может начать собственную передачу и тем самым нарушить процесс приема устройством C пакета от устройства B. Для борьбы с этим нежелательным явлением применяется способ, аналогичный используемому RS232: устройство посылающее данные уведомляет об этом остальные устройства при помощи специального короткого служебного пакета, а устройство которое готовится данные принять так же извещает остальные устройства коротким пакетом (в RS232 для этого используется выставление определенных уровней на специальных проводах). В нашем примере устройство A заметив маркер готовности к приему, посланный устройством C отложит посылку собственного пакета на время, необходимое для передачи устройством B.

Из выше сказанного видно что существенно упростить работу беспроводной сети может наличие выделенного узла координатора (подобно тому, как это имеет место в USB). Если речь идет о беспроводных локальных сетях, то такую роль обычно берет на себя узел, являющийся точкой доступа в Интернет. Очевидно производителям сетевого оборудования хочется продать подобные специализированные устройства, но так же очевидно и то, что потребитель хочет иметь возможность использовать и абсолютно однородную сеть из нескольких компьютеров.

Фактором усложняющим устройство оборудования для высокоскоростной передачи является еще и явление отражение СВЧ сигнала от многими предметами, свойственное так же и передача в стандарте GSM. Борьба с этим явлением требует дополнительного компонента - эквалайзера, предназначенного выделения исходного сигнала из многократно отраженного.

В довершении сказанного следует отметить необходимость защиты в беспроводной сети, где вероятность вторжения “непрошенных гостей” может быть существенно выше чем в обычной проводной, где проблема защиты информации уже стоит достаточно остро. Для компьютеров, соединенных WLAN эта проблема может решаться обычными сетевыми средствами, для более простых устройств защита может осуществляться в самом беспроводном протоколе.

BlueTooth

Протокол BlueTooth по своей сути является беспроводной заменой обычным последовательным шинам. Поэтому он в целом достаточно сильно похож на USB. BlueTooth соединяет пары устройств одно с другим, но не организует локальную сеть как таковую. По оценкам экспертов протоколы WLAN не являются конкурирующими по отношению к нему. Похоже это действительно так (хотя перед тем как безоговорочно принять эту точку зрения стоит вспомнить аналогичные неоправдавшиеся оценки о конкурирующем характере по

отношению друг к другу USB и FireWire).

Поскольку Bluetooth предназначен в частности для осуществления связи между самыми простыми устройствами, такими например как беспроводная гарнитура для мобильного телефона (и потенциально даже настенный выключатель), криптозащита реализована как часть самого протокола и это избавляет от необходимости реализовывать ее на прикладном уровне. Устройства могут запоминать идентификаторы друг друга и пароли доступа, таким образом формируется Privete Area network (PAN).

Самым нижним уровнем протокола является Baseband, для передачи используется метод прыгающей частоты, используемый диапазон 2.4GHz. Два различных типа физических соединений Synchronous Connection-Oriented (SCO) и Asynchronous Connections Link (ACL) используются для передачи звуковых потоков и данныхся соответственно. Передача звука происходит обособленным от данных способом, вероятно поэтому BlueTooth часто ассоциируется с беспроводными гарнитурами для мобильных телефонов, хотя очевидно что для управления функциями телефона, такими как поднятие трубки, используются и другие возможности протокола.

Выше находится Link Manager protocol (LMP), отвечающий за установление соединения и реализующий функции криптозащиты. При необходимости могут использоваться и нешифрованные соединения. Передача звука осуществляет с использованием названных двух нижних уровней. В принципе пользовательские приложения могут использовать и другие уровни BlueTooth, но стандартные устройства для простоты используют именно этот.

Следующим уровнем является Logical Link Control and Adaptation Protocol (L2CAP), этот протокол обеспечивает передачу пакетов и предназначен для использования протоколами более высокого уровня. Предполагается что разработчики могут использовать его для реализации собственного функционала. В версии BlueTooth 1.0 L2CAP использует только соединения типа ACL. Таким образом передача звука стандартным способом происходит полностью в обход L2CAP.

В свою очередь поверх уровня L2CAP уже работают протоколы более прикладного назначения:

Service Discovery Protocol (SDP), позволяющий обнаруживать и определять тип сервисов, предоставляемых другими устройствами.

Telephony Control Protocol, предназначенный для управления функциями телефона и факса, включая AT-команды.

RFCOMM – используется в качестве замены последовательного порта R232.

Поверх RFCOMM уже работают PPP и TCP/IP а так же UDP и протокол OBEX. OBEX предназначен для управления объектами в модели клиент-сервер, он так же предоставляет доступ к директориям для поиска нужных объектов по аналогии с LDAP. Сам по себе OBEX не является чем-т специфичным для BlueTooth, и применяется в частности вместе с IrDA. OBEX может использоваться например для обмена сообщениями vCard, которыми пользуется например программа Microsoft Outlook для обмена адресными книгами, что очень удобно для синхронизации адресов в компьютере с адресами в мобильно телефоне. RFCOMM так же используется для реализации целого набора прикладных протоколов, способ реализации каждого такого протокола называется профайлом, множество таких профайлов постоянно расширяется.

Поэксперементировать с уровнями протокола BlueTooth можно при помощи мобильного телефона Nokia 6600, в котором производитель обеспечил достаточно удобные API в среде Symbian и J2ME, или при при помощи специальных драйверов и библиотек в Linux или тулкитов в Windows.

WLAN

Беспроводные сети WLAN в отличие от BlueTooth предназначены быть аналогом обычных сетей LAN, а не последовательных шин и портов. В связи с этим они как правило имеют архитектуру существенно более общего назначения. Тем не менее для большей универсальности WLAN так же могут обеспечивать некотурую криптозащиту, хотя вобщем-то полагается что таковая восновном должна осуществляться пртоколами прикладного уровня, использующимися в обычных проводных сетях. Для передачи используются диапазоны частот 2.4GHz и 5GHz, в последних реализациях скорость передачи может достигать 11Mbit/sec. Такие сети, использующие пакетов типа Frame 802.11 получили название RadioEthernet. Несмотря на наличие стандарта часто возникают проблемы совместимости между устройствами, сделанными различными производителями, поэтому в последнее время ведущие производители оборудования в составе организации под названием Wireless Ethernet Compatability Alliance выработали дополнительные спецификации, уточняющие требования к оборудованию WLAN. Результатом этой работы стала Wi-Fi (Wireless Fidelity), эта технология является обязательной частью реализации новой платформы Intel для мобильных компьютеров под названием Centrino. Это обстоятельство должно послужить причиной быстрого продвижения Wi-Fi на рынке, поскольку сама платформа Centrino обладает важными характеристиками, которые делают ее полезной для применения в компьютерах типа Notebook, а именно высокая экономичность в отношении используемой электроэнергии, достигаемой за счет расширенного управления системой, позволяющего отключить или перевести в экономичный режим неиспользуемое в данный момент оборудование.

Литература

[1]“FireWire System Architecture” MindShare Inc. Addison-Wesley. ISBN 0201485354 1998

[2]Universal Serial Bus specification Ver.1.0-1.1-2.0 (electronic publication)

[3] “Смарт-карты, настольная книга разработчика” Тимоти М Юргенсен Скотт Б Гатери “Кудиц-Образ” ISBN 5-93378-069-3 2003

[4] “Помехоустойчивое кодирование” С И Самойленко “Наука” Москва 1966

[5] “Линейные последовательные машины” Р Г Фараджиев “Советское радио” 1975

[6] “Введение в QNX Neutrino(TM) 2” Роб Кертен ООО “Издательство “Петрополис”” ISBN 5-94656-025-9 Санкт-Питербург 2001

[6] “Java Card ™ Technology for Smart Cards Architecture and Programmer's Guide Foreword by Partice Peyrent ” ISBN 0201703297 Addison-Wesley 2000